VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroj pro generování syntetických dat fyzického návrhu databáze

A Tool for Generating Syntetic Data from a Physical Database Design

2018 Oskar Walder

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Oskar Walder**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Nástroj pro generování synetických dat fyzického návrhu databáze
A Tool for Generating Synetic Data from a Physical Database Design

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit nástroj pro generování velkého množství testovacích dat určených pro naplnění databáze.
Nástroj umožní:
1. Přehled existujících řešení pro generování dat.
2. Automatické rozpoznání a generování dat na základě analýzy struktury tabulek databáze.
3. Rozšiřování o vlastní vzory generování pro specifické struktury databáze.
4. Generování cizích klíčů pro propojené tabulky.
5. Definice skriptu pro generování specifických hodnot.

Práce bude obsahovat:
1. Přehled existujících řešení pro generování dat.
2. Implementaci výše popsané funkcionality.
3. Experimenty a vyhodnocení výsledků nad několika schématy OpenSource projektů.
4. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

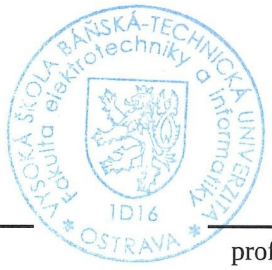
Dále dle pokynů vedoucího práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Čestné prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovat samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2018 ..... Walden .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, hlavně svému vedoucímu práce panu Ing. David Ježek, Ph.D. za podporu během zpracování této práce.

Abstrakt

Cílem této práce je navržení univerzálního generátoru dat, sloužící pro vytvoření fiktivních dat splňujících požadavky testované databáze nebo systému využívající perzistentního uložení dat v podobě databázového serveru. Během vývoje je cílem dodržet správných metod a metodik vývoje softwaru, použití vhodných navrhových vzorů, principů testování a celkového zhodnocení navrženého řešení. V úvodu je popsána problematika testování databázi a systému využívající databázového serveru pro perzistentní uložení dat. Dále jsou v úvodu popsány možnosti generování dat pro účely testování databáze, srovnání existujících řešení generování dat a nástin navrženého řešení popsané v této práci. V následující hlavní části práce je popsáno navržené řešení spolu s detailním popisem problémových částí. V předposlední částí je popsáno navržené řešení testování a verifikace celkového díla. V závěru práce je zhodnocení navrženého řešení, jeho využití a popsání dalších možností rozšíření a vývoje.

Klíčová slova: generování dat, umělá data, databáze, databázové testování

Abstract

The aim of this work is to design universal data generorator for creation of synthethic data for purpose of database testing and software using database server as persistent layer. During development should be used proper method of development, correct use of design patterns, testing a evaluation of designed solution. Introduction outlines the problem of database testing and testing of system based on database persisten layer. In introduction is also described current options of generation data for database testing. The following describes solution and overview of designed solution in this work. The following main part which describe designed solution with detailed description of problem sections. The last but one part describes developed solution for testing and validation of complete solution. The conclusion evaluates developed solution, use of solution and describes further extension and development.

Key Words: data generation, synthethic data, databse, database testing

Obsah

Seznam použitých zkratek a symbolů

WPF	–	Windows Presentation Foundation
MVVM	–	Model View ViewModel
UML	–	Unified Modeling Language
ASCII	–	American Standard Code for Information Interchange
SQL	–	Structured Query Language
CLR	–	Common Language Runtime
DLR	–	Dynamic Language Runtime
JIT	–	Just In Time
CSV	–	Comma Separated Values
RBG	–	Random bit generator
PRNG	–	Pseudorandom number generator

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Při návrhu informačního systému nebo jiného softwarového díla využívající databáze jako perzistentní vrstvy, je kladen velký důraz na efektivní návrh databázové struktury. Nesprávným návrhem se může efektivita přístupu k datům zhoršit až několikanásobně. Moderní databázové systémy nabízejí velké množství možností jak využití databáze a daného návrhu vhodně optimalizovat. Příkladem optimalizace může být vhodné použití databázových struktur reprezentující jednotlivé databázové tabulky, využití indexů nebo nastavení velikosti stránek při stránkování databázovým systémem. Při velkém počtu možností nastavení musí i zkušený uživatel navrhovaný systém před finálním nasazením důkladně a opakovaně otestovat. Pro účely testování je třeba databázi naplnit testovacími daty, která se svými vlastnostmi co možná nejvíce přibližují skutečným datům, protože jenom v takovém případě je možné u daného systému co nejpřesněji nasimulovat skutečné využití daného návrhu a odhalit tak případné chyby v návrhu či nastavení databázovém systému. Díky tomuto požadavku je většina uživatelů nucená tyto fiktivní data vytvořit pomocí skriptu, nebo jiného vlastnoručně vytvořeného programu.

Vývoj programu řešící generování fiktivních dat pro účely testování jednoho specifického návrhu databáze může být náročný a zbytečně se opakující proces. Pro odstranění tohoto opakujícího se procesu by bylo ideální navrhnout generátor dat, který by se dal univerzálně aplikovat na libovolný databázový návrh. Pro splnění požadavku na univerzální generátor dat, jsem vytvořil řešení generování dat využívající znalosti návrhu databáze, pro nichž je třeba daná data vytvořit. Řešení je možné využít pro univerzální generování dat různých databázových návrhů spolu s aplikací rozsáhlých možností nastavení a omezení na generovaná data. Toto řešení funguje na základě načtení informací o návrhu databáze ze systémového katalogu dané databáze a následného generování za pomocí univerzálních generátorů datových typů, vytvořených datových sad, nastavených vlastností a omezení generování, anebo doprogramování požadované funkcionality za pomocí Python skriptu.

Při návrhu byl kladen důraz na dodržení správných metodik vývoje softwaru, jako je vhodné využití návrhových vzorů, principů testování, umožnění snadného rozšíření softwaru a také návrh uživatelsky přívětivého rozhraní. Úvodní část dokumentu popisuje princip fungování generování dat a návrh celého řešení. V následující části je popsáno řešení vzniklých problémů při návrhu řešení univerzálního generování dat. V závěru je zhodnoceno řešení z pohledu funkcionality, využitelnosti a možností dalšího rozšíření díla.

2 Porovnání dostupných možností

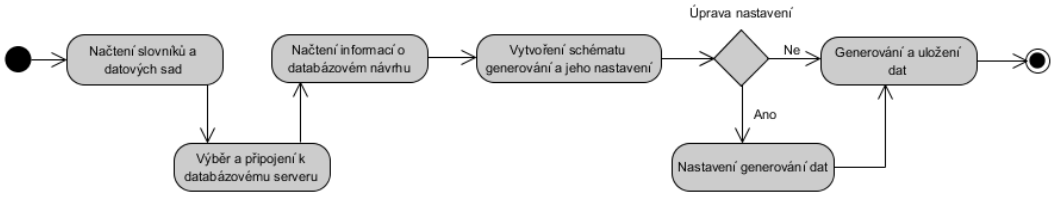
Na trhu je dostupná poměrně velká sada nástrojů schopná generovat fiktivní data pro účely testování a splňující požadované vlastnosti integritního omezení testované databáze. Od jednodušších nástrojů sloužící pro naplnění malých databázových schémat až po složitější nástroje snažící se pokrýt obsáhlejší databázové schémata a složité požadavky na vlastnosti generovaných dat. Mezi nejpoužívanější datové generátory patří DTM Data Generator, RedGate Data Generátor, anebo Datový generátor prémiové verze vývojového studia s názvem Visual Studio Premium. I když většina zmíněných nástrojů obstojně splňuje požadavky na vlastnosti dat, tak nastává velký problém s dostupností těchto nástrojů a jejich rozšiřitelností v případě chybějící funkcionality. Ve všech třech zmíněných případech se jedná o proprietární software daných firem, a tak ani jeden ze zmíněných nástrojů není poskytován bezplatně. I přesto, že zmíněné nástroje patří mezi nejpoužívanější, za vývojem stojí velké podniky s početnými týmy vývojářů, tak nedokážou tyto nástroje vytvořit data, která by se hodila pro všechny navržená schémata databáze se všemi existujícími možnými požadavky na vlastnosti generovaných dat.

V praxi ani není možné vytvořit nástroj pro generování všech možných požadavků na nastavení a vlastnosti generovaných dat. Dalšími nedostatky můžou být i nevhodně navržené nebo zvolené možnosti generování, které byly realizovány ve zdrojovém kódu daného nástroje. Příkladem těchto nedokonalostí může být fakt, že i přes znalost návrhu databázového schématu pro která májí být příslušná data generována, nedokáži některé z těchto nástrojů vhodně rozpoznat propojení tabulek pomocí cizích klíčů a tyto klíče vytvořit, tak aby splňovala referenční integritu navržené databáze. Jelikož tyto nástroje nejsou poskytovány pod Open Source licencemi a neposkytují žádnou možnost rozšíření programu o potřebnou funkcionalitu, ať už za pomocí úpravy zdrojového kódu nebo skriptového rozhraní, nezbývá vývojářům než pracně vyvinout vlastní generátor dat splňující potřebnou funkcionalitu. Jelikož je třeba pro účely testování mít dostupná testovací data co možná nejdříve v průběhu procesu vývoje databázového systému, potom implementace takového generátoru není rychlou, a tudíž vhodnou možností. Tým musí přidělit jedince či skupinu vývojářů a testerů pro práci na úkolu vývoje generátoru dat na místo toho, aby se mohli tito jedinci podílet na úkolu vývoje cílového softwarového díla.

Proto se řešení popsané v této práci snaží poskytnout nástroj pro generování potřebných dat, které bude díky správnému návrhu, zpřístupněnému skriptovacímu rozhraní a otevřeném zdrojovém kódu co nejlépe využitelné a snadno rozšiřitelné. Řešení se nepokouší pokrýt všechna možná potřebná možnosti generování ani funkcionalitou překonat výše zmíněné nástroje, ale cílem je poskytnout vývojářům alespoň částečnou podporou v případě nutnosti vývoje vlastního generátoru dat. Tato podpora může být v možnosti využití knihovny sloužící pro generování dat, v možnosti před vytvoření si dat za pomocí řešení popsané v této práci, uložení do souboru a následné dodatečné úpravě za pomocí již jednoduchého skriptu.

3 Základní popis navrženého řešení

Základní myšlenka navrženého řešení je možnost automatického vytvoření generujících struktur, jejich příslušné nastavení a umožnění tak jednoduše vytvářet vzorky dat vhodné pro naplnění testované databáze, bez nutnosti vývoje programu generující požadovaných vzorků dat. Pro potřeby rozeznání požadovaných vlastností generovaných vzorků dat musí mít řešení možnost přístupu ke zdroji obsahující informace o cílovém umístění generovaných dat, anebo musí uživatel příslušné nastavení provézt manuálně. Vhodným zdrojem informací pro nastavení generování dat v případě požadavků na naplnění databáze testovacími daty, je příslušný systémový katalog databáze. Systémový katalog databáze je nutný pro správný běh každého databázového serveru a poskytuje informace o navrženém schématu databáze, příslušných tabulkách, datových typech atributů tabulek a mnohem více. I přes velké množství uložených informací v systémovém katalogu neobsahuje systémový katalog všechny informace o požadovaném nastavení pro generování dat, a proto je potřeba do procesu nastavení generování dat zapojit i přístup uživatele. Příkladem může být atribut tabulky s datovým typem VARCHAR a názvem "krestnijmeno". V takovém případě systém obsahuje informaci o hodnotách, které se mohou v datového typu VARCHAR vyskytoval, ale nevyžaduje ani nemá informaci o tom, že skutečně vložené hodnoty by měly obsahovat řetězce křestních jmen. Databázový systém takovou informaci ke své správné funkci ani nepotřebuje, a tak nemá smysl, aby takovou informaci evidoval. Z tohoto důvodu je zapotřebí do procesu nastavení generování zapojit i uživatele, který upraví zbylé požadované nastavení. K tomuto zapojení do procesu slouží navržená aplikace poskytující uživatelské rozhraní a využívající knihovnu generování dat. Uživatel může pomocí uživatelského rozhraní nastavit příslušná omezení, či zvolit generování dat na základě datových sad a slovníků. Jakmile jsou správně vytvořeny a nastaveny požadované struktury generátoru, pak je možné vygenerování vzorků dat a jejich následné uložení. Pro uložení je možno využít připojení k databázi a vytvořená data uložit do databázového serveru, anebo využít lokálního uložení do datového souboru pro další potřeby zpracování uživatelem.



Obrázek 1: Zjednodušený aktivitní diagram procesu generování dat

Navržené řešení je rozděleno do tří softwarových modulů, kde se každý modul zaměřuje na právě jednu problematiku řešení.

Prvním a základním popsaným modulem je knihovna poskytující funkcionalitu generování dat s spolu se všemi ostatními potřebnými funkcemi modelu generování dat. Knihovna je de-

komponována do několika funkčních bloků pomocí definovaných jmenných prostorů. Každý blok je zaměřen na právě jednu z problematik generování dat v navrženém řešení. Jedním z prvních úkolů knihovny je připojení k databázi, pro niž budou data generována a zjistit veškeré možné informace podle kterých bude možno sestavit schéma generování splňující požadavky integritního omezení dané databáze. Jakmile knihovna obdrží veškeré potřebné fragmenty informací o navržené databázi, potom sloučí tyto fragmenty do celku, který popisuje schéma a proces generování dat pro naplnění dané databáze vygenerovanými daty. Mezi informace, které tvoří schéma generování dat patří informace o počtu tabulek, nastavení atributů jednotlivých tabulek, provázání tabulek pomocí primárních klíčů a další informace o integritních omezení zvolené databáze. Jakmile knihovna sestaví úplný model schématu generování dat, je možno tento model analyzovat a identifikovat případné vzory ve schématu. Vzorem ve schématu generování dat se myslí určitá část schématu, která se často opakuje v různých schématech návrhu databáze a obsahuje stejné požadavky na generování dat. Příkladem jednoho s nejčastěji užívaného vzoru může být tabulka "účet"s atributy "jméno", "příjmení"nebo s libovolnými jinými synonymy popisující stejný vzor. Knihovna poskytuje možnost nastavení různých zdrojů generátorů pseudonáhodných čísel spolu s různými vlastnostmi distribučního rozdělení a omezení. Knihovna dále poskytuje velkou sadu možností nastavení a omezení generování dat, jakými jsou například možnost generování unikátních sekvencí čísel, řetězců, datumů, nebo vytvoření vazebních klíčů splňující různé vlastnosti provázaní mezi jednotlivými tabulkami. Posledním blokem je blok poskytující implementaci skriptového rozhraní. Toto skriptové rozhraní slouží pro nastavení potřebné chybějící funkcionality, jenž není poskytnuta v knihovně generování dat, za pomocí vloženého skriptu do skriptového rozhraní.

Druhým modulem je aplikace poskytující uživatelské rozhraní, zpřístupňující možnost dodatečného nastavení vlastností generovaných dat, vizualizaci načtených informací o databázi a ovládání generování dat za pomocí komponent uživatelského rozhraní. Za pomocí aplikace může uživatel upravit již před vytvořené generující struktury, jejich pořadí generování, nebo vložit, odladit a validovat chybějící funkcionalitu pomocí Python skriptu do skriptového rozhraní.

Posledním modulem je projekt obsahující sadu unit testů pro testování správné funkcionality generování dat obsaženém v knihovně pro zpracování a generování dat. Sada testů ověřuje správnost základních funkcí, správnost generování zvolených distribučních funkcí a ostatních omezení generování.

4 Knihovna pro generování dat RealDataGenLib

Pro oddělení doménové logiky od prezentační vrstvy byla veškerá funkcionalita generování implementována v knihovně generování dat. Aplikace následně využívá tuto knihovnu pro potřeby generování dat. Knihovna obsahuje funkcionalitu pro generování dat, funkcionalitu pro generování náhodných čísel založené na různých typech pseudonáhodných generátorů, funkcionalitu pro generování distribučních funkcí, přístup k databázovému serveru a systémového katalogu. Architektura knihovny byla navržena s důrazem na správné použití návrhových vzorů, metodik a metod pro udržení největší nezávislosti mezi jednotlivými softwarovými komponenty.

Funkcionality knihovny by se dala sloučit do čtyřech fázích, které se v běžném případě vykonávají postupně jedna za druhou:

1. Získání informací o fyzickém návrhu databáze z databázového serveru
2. Automatické nastavení generování na základě znalosti návrhu databáze
3. Manuální dodatečné nastavení generování pomocí požadavků uživatele vložené pomocí uživatelského rozhraní
4. Generování dat a jejich následné uložení

4.1 Získaní fyzického návrhu databáze

Pro potřeby navržení struktury generovaných dat a jejich vhodného nastavení, je potřeba získat informace o fyzickém návrhu databáze pro niž mají být data generována. Pro správné nastavení generování potřebuje řešení získat informace o složení databázových tabulek, definice jednotlivých atributů, složení primárních a cizích klíčů, integritním omezením, vzájemné provázání tabulek a dané nastavení vazeb. Za základě množství získaných informací o fyzickém návrhu dokáže řešení lépe nebo hůře přizpůsobit proces generování dat pro potřeby cílového schématu testované databáze.

4.1.1 Nejednotná definice systémového katalogu databázových serverů

Jelikož databázový systém potřebuje pro svůj správný běh znát všechny definice databázových tabulek a sloupců jsou potřebná data pro generování přístupná v systémovém katalogu databáze. Systémový katalog databázového serveru obsahuje informace o jednotlivých databázích, uložených tabulkách, vlastnosti jednotlivých sloupců, informace o uživatelích jejich přístupových právech a další v podobě databázových tabule. Získávání dat tedy probíhá v podobě načtení dat ze správných systémových tabulek. Bohužel neexistuje jednotná definice, jak by měli být systémové informace uloženy v tabulkách, ani jak by se tyto tabulky měli jmenovat, proto každý tvůrce databázového systému si volí vlastní strukturu systémového katalogu. Z tohoto důvodu

Tabulka 1: Popis jmenných prostorů knihovny

Jmenný prostor	Popis
Database	Jmenný prostor poskytující definici rozhraní a funkcionalitu přístupu k databázi.
Database.CatalogDef	Jmenný prostor definující obecné třídy pro požadované informace o databázovém návrhu.
GenEngine	Obecný jmenný prostor generátoru dat.
GenEngine.Datatype	Jmenný prostor obsahující implementaci generátoru dat pro specifické databázové datové typy.
GenEngine.DataStructure	Obsahuje implementaci struktur pro správu pořadí generování, porovnání stromu fyzického návrhu a identifikaci vzorů fyzického návrhu.
GenEngine.Randomness	Obsahuje implementaci generátoru pseudonáhodných čísel založených na lokálních i vzdálených servisních metodách.
GenEngine.Randomness.Distribution	Jmenný prostor obsahující implementaci distribučních funkcí.
GenEngine.Scripting	Jmenný prostor poskytující funkcionalitu skriptování.
Services	Jmenný prostor pro implementaci rozhraní k externím službám.
Services.QuantumRandom	Implementace načítání náhodných čísel z externího kvantového generátoru.

není možné definovat uniformní metody přístupu k databázím, které by byly použitelné bez jakékoliv modifikace u rozdílných tvůrců databázových systému.

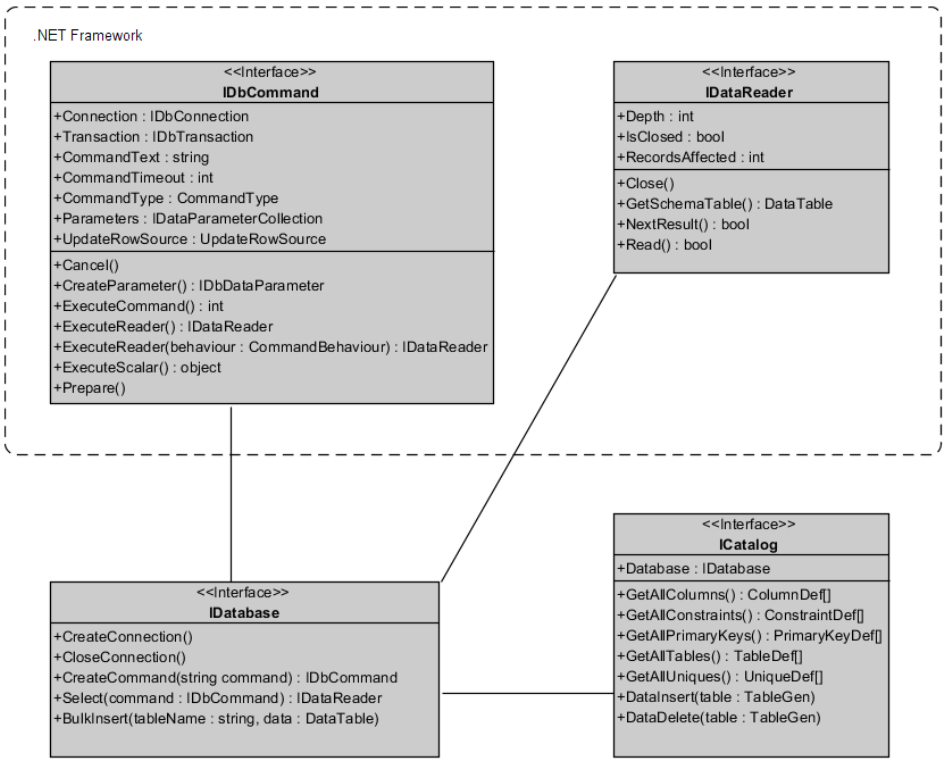
4.1.2 Obecné rozhraní pro přístup k databázi a systémovému katalogu databáze

Aby bylo možno přistupovat k různým databázovým serverům a přidávat databázové servery od jiných tvůrců bez modifikace architektury softwaru, pracuje aplikace s předem definovaným jednotným rozhraním. Pro každý přidaný databázový server musí platit správná implementace a funkce tohoto rozhraní. Přístup k databázovému serveru musí implementovat dvě rozhraní, první rozhraní s názvem IDatabase slouží k vytvoření připojení k databázi, ukončení připojení k databázi a k jednotlivým operacím nad databázovými tabulkami. Druhé rozraní ICatalog slouží k načítání užitečných dat ze systémového katalogu. Návrh dvou rozhraní pro jeden databázový server je z důvodu existence různých verzí databázového serveru jednoho tvůrce databáze. Bez použití dvou rozhraní by nebylo možno vhodně rozlišit a rozdělit implementace přístupu k databázovému serveru a jeho katalogu u různých verzí databázového serveru s různou definicí systémových tabulek.

Framework .NET obsahuje sadu implementací přístupu k databázových serverů od různých tvůrců databází. Všechny tyto implementace jsou vytvořeny na základě realizace uniformních

rozhraní definovaných ve jmenném prostoru System.Data .NET frameworku, proto je možné pro definici obecného rozhraní přístupu k databázi využít těchto uniformních rozhraní s jistým vědomím toho, že bude možné za pomocí obecného rozhraní implementovat přístupy k různým databázovým serverům a splnit požadavek na jednotné rozhraní přístupu k databázím.

Jmenný prostor System.Data obsahuje mimo jiné definice rozhraní IDbCommand a IDataReader, které jsou využity v definici rozhraní obecného přístupu k databázi. IDbCommand definuje předpis rozhraní pro instanci dotazovacího příkazu SQL pro databázový server a IDataReader definuje obecné rozhraní pro zpracování výstupu typu DataTable z vykonaného příkazu rozhraní IDbCommand.



Obrázek 2: Třídní diagram definice obecného rozhraní přístupu k databázovému serveru

Jakmile aplikace obdrží za pomocí implementovaného rozhraní pro zvolený databázový systém všechna potřebná data, tak tyto data vhodně sloučí do jednotné definice sloupce třídy ColumDef, obsahující všechny potřebné informace pro další zpracování.

Poslední potřebnou metodou obecného rozhraní přístupu k databázi je metoda mapování instance třídy typu ColumnDef na instanci typu ColumnGen. Různé databázové systémy definují pro stejné nebo podobné datové typy rozdílné názvy. Příkladem může být v MSSQL databázovém systému reprezentace řetězce pomocí klíčového slova VARCHAR na rozdíl od Oracle databázového systému, kde je použito klíčové slovo VARCHAR2 pro stejný datový typ. Za po-

Tabulka 2: Popis funkcí rozhraní pro přístup do databázového serveru

Název funkce	Popis
OpenConnection	Vytvoří a otevře připojení k databázovému serveru na základě řetězce obsahující informace o připojení.
CloseConnection	Vynutí ukončení připojení k databázovému serveru bez ohledu a na počet otevřených připojení.
CreateCommand	Vytvoří instanci příkazu pro databázový server na základě řetězce obsahující příkaz jazyka SQL.
GetAllTables	Načte z databázového serveru informace o všech tabulkách, jejich názvech, zařazení do databázových schémat a navrátí v podobě zřetězeného seznamu třídy TableDef.
GetAllColumns	Načte z databázového serveru informace o všech sloupcích, jejich názvech, zařazení do databázových tabulek, vlastnosti jako datový typ nebo maximální délka a navrátí v podobě zřetězeného seznamu třídy ColumnDef.
GetAllConstrains	Načte z databázového serveru informace o propojení jednotlivých databázových tabulkách, zdrojových tabulkách, cílových tabulkách a navrátí v podobě zřetězeného seznamu třídy ConstrainDef.
GetAllPrimaryKeys	Načte z databázového serveru informace o primárních klíčích, jejich popisech, názvech sloupců, zařazení do databázových schémat a navrátí všechny tyto informace v podobě zřetězeného seznamu PrimaryKeyDef.

mocí této funkce je řešení schopno zvolit vhodný generátor datového typu pro jednotlivé datové typy databázového systému.

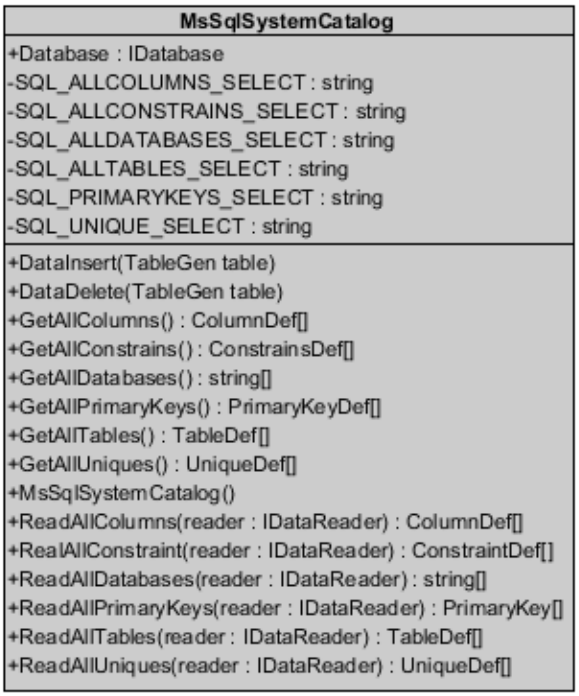
4.1.3 Realizace databázového rozhraní pro MSSQL databázi

Aplikace obsahuje implementaci pro databázový server firmy Microsoft s názvem Microsoft SQL Server ve verzi 2014. Systémový katalog této databáze obsahuje informace o uživatelských databázových tabulkách a sloupcích v systémových tabulkách.

Použité tabulky systémového katalogu:

1. INFORMATION_SCHEMA.TABLES obsahující informace o databázových tabulkách
2. INFORMATION_SCHEMA.COLUMNS obsahující informace o atributech jednotlivých tabulek
3. SYS.INDEXES obsahující informace o indexaci jednotlivých primárních klíčů tabulek
4. INFORMATION_SCHEMA.TABLE_CONSTRAINTS obsahující informace o propojení tabulek za pomocí primárních a cizích klíčů.

Díky těmto výše zmíněných informacím jsme schopni sestavit úplný model struktury navržené databáze v paměti programu, provést nad touto strukturou analýzu a připravit nastavení pro generování příslušných dat.



Obrázek 3: Návrh třídy pro získávání dat z MSSQL databázového serveru

Třída obsahuje dva typy naimplementovaných funkcí. Prvním typem jsou funkce splňující realizaci obecného rozhraní přístupu k databázi v podobě vytvoření příkazu pro databázový server, jeho spuštění a získání navraceného výsledku. Druhým typem funkcí jsou funkce pro zpracování navrácených výsledků ze spuštěných příkazů. Všechny tyto funkce přijímají na vstupu instance třídy IDataReader, která obsahuje informace o navrácené datové sadě a na výstupu vracejí vytvořené seznamy požadovaných datových typů. Kromě zmíněných dvou typů funkcí, třída dále obsahuje řetězce reprezentující SQL příkazy pro získaní požadovaných informací z databázového serveru. Jelikož tyto příkazy se skládají z kombinace SQL příkazů tvořící dlouhý a víceřádkový řetězec, z toho důvodu byly přesunuty do externích zdrojů knihovny a jsou pouze odkazovány v této třídě.

První použitou funkcí rozhraní přístupu k databázovém systému je funkce s názvem GetAllDatabases. Tato funkce navrací seznam řetězců obsahující názvy dostupných databází na zvoleném databázovém serveru. Funkce načítá informace ze systémové tabulky s názvem sys.databasys. Tato tabulka obsahuje vždy jeden záznam pro jednu databázovou instanci serveru. Mezi potřebné atributy záznamu patří atribut „name“ datového typu varchar, který obsahuje název databáze. Druhým užitečným atributem je atribut „dbid“ typu int, který obsahuje unikátní identifikátor databázové instance. Identifikátor instance databáze je důležitý z důvodu rozlišení systémových

databází od uživatelských, prvních 6 instancí obsahuje totiž systémové databáze a ty uživatelské začínají až identifikátorem s hodnotou 7. Sestavením jednoduchého dotazu v dotazovacím jazyce SQL získáme seznam všech uživatelských databází. Tato funkce je použita k vytvoření nabídky dostupných databázi pro výběr po zadání IP adresy serveru.

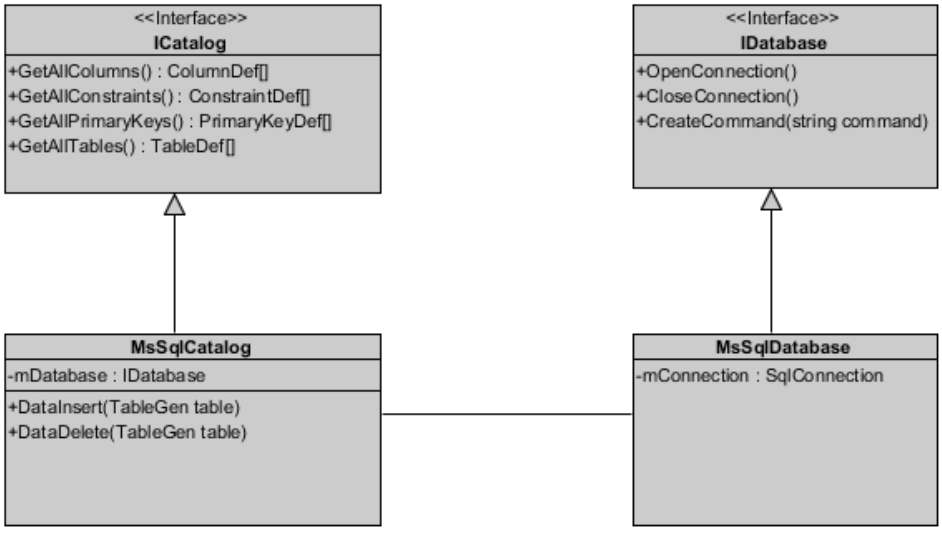
Metoda GetAllColumns načítá informace o všech použitých atributech v jednotlivých tabulkách. Mezi tyto informace patří názvy atributů, jejich datové typy, omezení délky a další nastavená omezení.

Metoda GetAllPrimaryKeys vrací seznam atributů, které jsou v databázi použity jako primární klíče. Tyto atributy obdrží během procesu generování dat požadovanou vlastnost na generování unikátních sekvencí, aby bylo splněno integritní omezení databáze na unikátní primární klíče.

Jelikož v databázovém serveru může být nastaven požadavek na unikátnost záznamů i u atributů, které nejsou primárními klíči, je potřeba získat i seznam ostatních unikátních atributů, k tomuto požadavku slouží metoda GetAllUniques.

Po obdržení všech informacích o atributech, je potřeba vhodně rozdělit celý seznam do reprezentace jednotlivých tabulek. Pro získaní informací jakým způsobem se mají všechny atributy rozdělit do tabulek existuje funkce GetAllTables. Tato funkce navrací seznam tabulek spolu s informacemi o jednotlivých tabulkách, jako je název tabulky, zařazení do schématu a složení atributů.

Jako posledním krokem se provede provázání jednotlivých tabulek do celkového schématu. Metoda GetAllConstrain navrací informace o způsobu, jakým se mají jednotlivé tabulky mezi sebou navzájem provázat.



Obrázek 4: Třídní diagram realizace rozhraní pro přístup do databáze

Program načítá informace o fyzické reprezentaci pomocí systémových katalogů jednotlivých

databází. Na třídním diagramu je návrh implementace přístupu do databáze MSSQL, pro přidání možnosti připojení k databázi jiného tvůrce je třeba implementovat navržené rozhraní přístupu do databáze.

Navržené řešení obsahuje sadu obecných generátorů datových typů, které pokrývají požadovanou funkcionalitu určených vlastností databázových datových typů. Všechny tyto generátory obsahují možnosti nastavení přizpůsobení tak, aby jeden typ byl schopen pokrýt i více databázových datových typů.

Řešení má vytvořené tyto obecné generátory:

1. IngGen – generátor celočíselných datových typů
2. StringGen – generátor řetězcových datových typů
3. DecGen – generátor reálných datových typů
4. DateTimeGen – generátor časových datových typů
5. DateTimeGen – generátor časových datových typů
6. DateGen – generátor datumových datových typů

Všechny tyto obecné generátory dědí z obecného předka typu ColumnGen, a tak kromě vlastních datových nastavení obsahují také obecně sdílené vlastnosti. Mezi tyto sdílené vlastnosti patří názvy sloupců, tabulek a schémat, do kterých jsou zařazeny. Dále také sdílené vlastnosti generování, jakými jsou: distribuční rozdělení, vazby na ostatní sloupce v podobě cizích klíčů nebo také prostor pro uložení vygenerovaných dat. Třída ColumnGen také definuje skupinu abstraktních funkcí, které musí jednotlivé generátory dat přetížit pro správnou funkcionalitu generování.

Jelikož řešení pracuje s obecnými generátory datových typů a databázový server využívá specifické datové typy, je třeba definovat jakým způsobem se budou jednotlivé datové typy reprezentovat v řešení generování dat. Pro tyto účely je třeba implementovat funkcionalitu definovaného obecného rozhraní a definovat mapování jednotlivých databázových datových typů na obecné generátory datových typů, a to takovým způsobem, že jeden nebo více databázových datových typů je reprezentován jediným obecným generátorem datového typu.

Dokumentace MSSQL databázového systému rozděluje datové typy do kategorií podle vlastností datových typů, a to do následujících skupin. [1]

Rozdělení datových typů do skupin podle dokumentace:

1. Přesné typy určující přesnou hodnotu datového typu.
2. Aproximované typy určující pouze přibližnou hodnotu datového typu.
3. Datové a časové typy definující čas, časová razítka anebo datumy.
4. Znakové řetězce pracující s řetězci ASCII znakové sady.

5. Unicode znakové řetězce pracující s řetězci Unicode znakové sady.

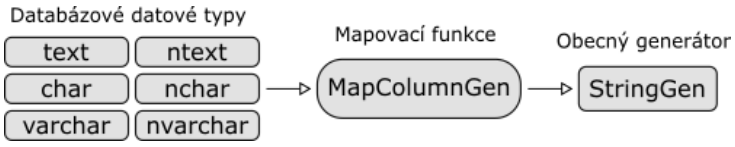
6. Binární řetězce

7. Ostatní datové typy

Takové rozdělení do skupin pro účely generování dat však není vhodné. Důvodem je, že generátor dat pracuje pouze se zjednodušeným modelem datových typů. Dalším z důvodů je také fakt, že řešení neimplementuje generátory dat pro binární řetězce, geometrické typy, XML typy a další. Řešení nerealizuje tyto generátory z důvodu nejasností obsahu těchto datových typů, jejich velkého množství možných generování a minimální využití v návrhu databázového systému.

Místo rozdělení do skupin na základě dokumentace bylo zvoleno vlastní rozdělení, které nahlíží na datové typy z pohledu generování a jejich skutečného využití. Skupiny pro mapování databázových datových typů na obecné generátory datových typů, byly vytvořeny následujícím způsobem.

Datové typy řetězců znaků byly shluknuty všechny do jedné skupiny. Toto je možné díky faktu, že generátor vytváří pouze znakové řetězce ASCII znakové sady a ty jsou přijímány i datovými typy pracující s Unicode znakovou sadou.

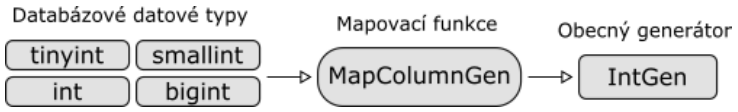


Obrázek 5: Mapování databázových datových typů na řetězcový generátor

Databázový systém MSSQL definuje dva základní řetězcové datové typy, CHAR, VARCHAR. Třetí datový typ TEXT je pouze ponechán z důvodu zpětné kompatibility se staršími verzemi databázového systému využívající tohoto typu.

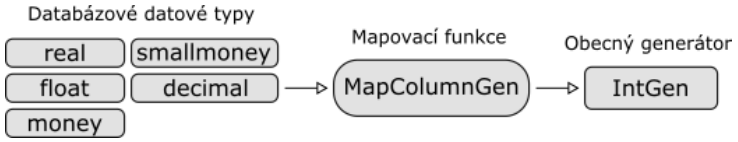
Datový typ VARCHAR je datovým typem pro uložení textových řetězců s proměnou délkou. Řetězec je reprezentován v paměti vždy na základě skutečné velikosti vloženého textu, a ne na základě nastavené maximální délky řetězce. U tohoto typu nedochází k plýtvaní paměťovým prostorem. Datový typ CHAR je datový typ prezentující textové řetězce stejné délky. I do tohoto typu lze uložit řetězce kratší délky, než je nastavená délka, ale v takovém případě dojde k plýtvaní místem v paměti, jelikož datový typ je prezentován vždy stejným počtem bytů odpovídající nastavené maximální velikosti. Výhodou použití tohoto typu může být větší efektivita zpracování než u datového typu VARCHAR. Tyto datové typu jsou dále rozšířeny o možnost uložení UNICODE znaků, a to ve verzích obsahující prefix „n“. Datové typy umožňující uložení UNICODE znaků jsou v paměti reprezentovanými dvakrát větším počtem bytů než u typů, ve kterých je možno ukládat pouze ASCI znaky.

Pro celočíselné datové typy definuje databázový server čtyři celočíselné datové typy, každý z nich se liší v maximální a minimální hodnotě, která je možná do datového typu vložit, a tudíž i



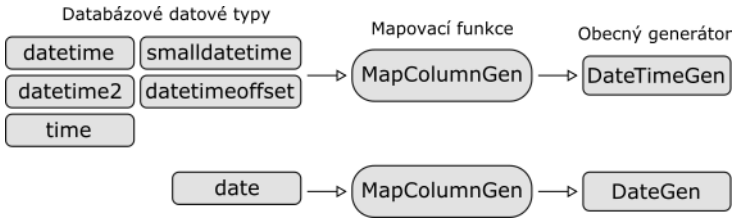
Obrázek 6: Mapování databázových datových typů na celočíselný generátor

velikosti reprezentace v paměti. Do datového typu tinyint je možno uložit 2^8 hodnot, do smallint 216 hodnot, do int potom 2^{32} hodnot a do bigint 2^{64} hodnot.



Obrázek 7: Mapování databázových datových typů na generátor reálných čísel

Pro reálná čísla definuje MSSQL databázový systém hned několik datových typů. Tyto typy se dají rozdělit mezi datové typy určující přesně hodnotu čísla a mezi přibližné datové typy určující pouze přibližnou hodnotu. Mezi datové typy určující přesnou hodnotu datového typu patří datové typy smallmoney, money a decimal. SmallMoney přesně určuje 2^{32} a Money 2^{64} hodnot posunutých tak, aby se daly definovat hodnoty na čtyři desetinná místa. Decimal oproti datových typů pro práci s měnami umožňuje volitelně definovat počet desetinných míst.



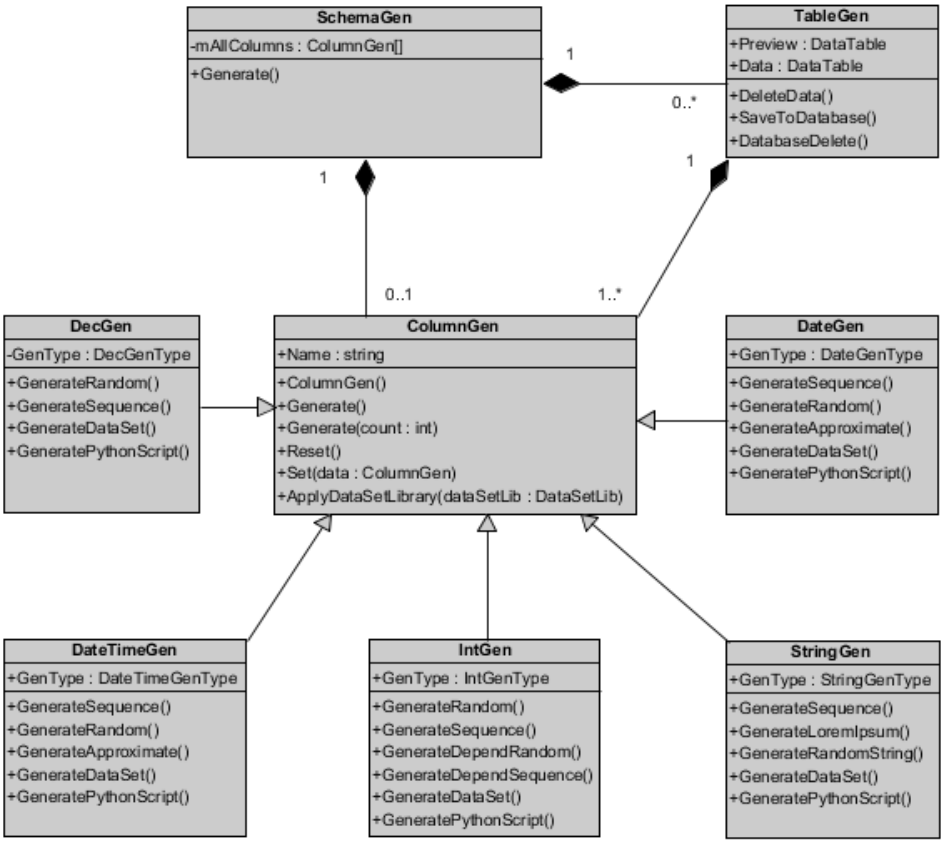
Obrázek 8: Mapování databázových datových typů na generátor datumů a časových razítek

Posledními dvěma obecnými datovými generátory jsou DateTimeGen pro generování časových razítek a DateGen pro generování datumů. Nastavení generátoru DateTimeGen se zaměřuje na generování časových razítek a času obecně na rozdíl od DateGen, který má za úkol pouze generovat datumy s určitým nastavením nebo z předem vytvořené datové sady.

4.2 Zpracování fyzického návrhu databáze

Jakmile jsou všechna data úspěšně načtena z databázového serveru, aplikace sloučí získané fragmenty informací do celku definovaného aplikací. Datový generátor pracuje s instancí třídy SchemaGen, která obsahuje strukturu s informacemi o jednotlivých tabulkách a jejich složení. K rozdělení došlo z důvodu potřeby rozlišení nastavení jednotlivých generátorů.

Třída SchemaGen popisuje získaný model navržené databáze z databázového serveru, složení tabulek ve schématu, nebo provázaní jednotlivých tabulek za pomocí primárních a cizích klíčů.



Obrázek 9: Obecný třídní diagram architektury datového generátoru

Třída také obsahuje informace týkající se o pořadí generování jednotlivých dat, strom fyzického návrhu databáze pro porovnání s identifikovanými vzory, pořadí vkládání do databázového serveru, pořadí generování navzájem nezávislých dat, strom generování na sobě závislých dat, pomocné metody pro sloučení fragmentů dat do schématu, metody pro roztřídění závislých a nezávislých dat a metody uživatelského rozhraní.

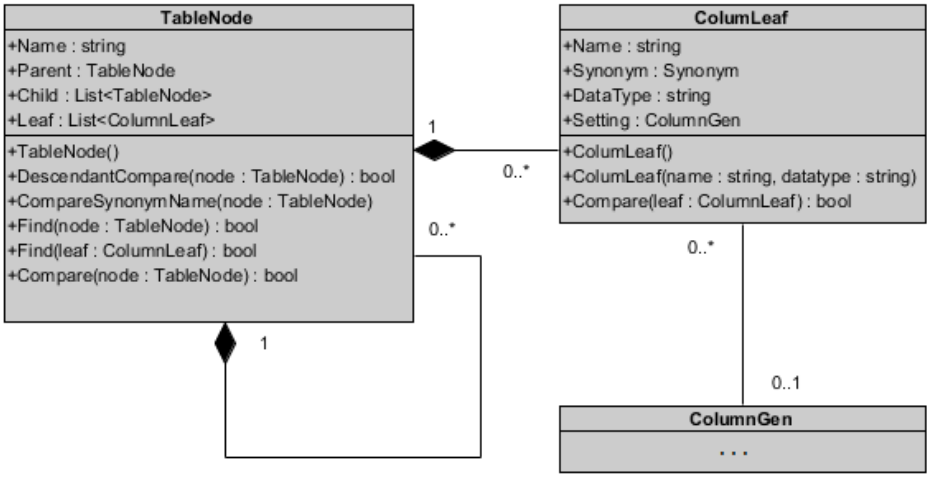
Třída TableGen obsahuje vlastnosti spjaté s databázovou tabulkou a vlastnosti generování. Mezi vlastnosti databázové tabulky patří složení jednotlivých databázových sloupců, název tabulky a zařazení tabulky do schématu. Mezi vlastnosti generátoru potom patří: požadovaný počet generovaných řádků, vygenerovaná data pro ukázku v uživatelském rozhraní a vygenerovaná data pro následné uložení do databázového serveru nebo do lokálních souborů.

Třída ColumnGen je obecnou třídou, která slouží jako předek pro jednotlivé datové typy sloupců, sdílí obecné vlastnosti a metody. Dále obsahuje předpis funkcí nutných pro přetížení v potomcích

4.2.1 Strom fyzického návrhu

Strom fyzického návrhu je stromová datová struktura sloužící k porovnání fyzického návrhu s přednastavenými vzory fyzického návrhu a aplikace nastavení nalezeného vzoru na daný strom fyzického návrhu. Dále slouží strom fyzického návrhu k vygenerování stromu obsahujícího pořadí generování jednotlivých sloupců a stromu určujícího pořadí vkládání tabulek do databáze.

Strom se skládá z uzlu obsahující odkaz na rodiče, ze seznamu odkazů na potomky. Dále uzel obsahuje seznam uloženého nastavení generování, které se má aplikovat na list atributů daného uzel v případě shody stromu fyzického návrhu s definovaným vzorem. Definice uzlu je uložená ve třídě TableNode, která zaobaluje veškerou práci se stromem a jednotlivými uzly.



Obrázek 10: Definice stromu fyzického návrhu databáze

Nejdůležitější funkcí uzlu stromu je porovnávání jednotlivých uzlů a uzlů složených do stromové struktury. Třída TableNode obsahuje hned několik funkcí pro porovnávaní obsažených objektu ve třídě.

Tabulka 3: Popis porovnávacích a vyhledávacích funkcí

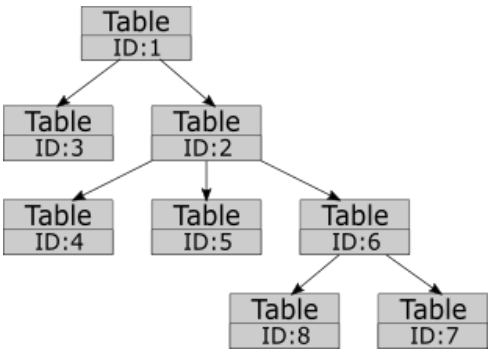
Název funkce	Popis
Compare	Zaobaluje porovnávací funkcionalitu a volá ostatní porovnávací metody.
CompareSynonymName	Porovnává názvy na základě nastavených synonym.
DescendantCompare	Porovnává potomky jednotlivých uzlů.
Find(TableNode)	Funkce vyhledává uzel v daném stromu.
Find(ColumnLeaf)	Funkce vyhledávající atribut tabulky v daném stromu.

Třída ColumnGen obsažená v návrhu stromu obsahuje nastavení generování, které se má aplikovat na strom v případě shody s definovaným vzorem. Samotný strom fyzického návrhu neobsahuje takovou informaci, nastavení je definováno pouze ve vzorech.

Třida ColumnLeaf obsahuje odkaz na třídu Synonymum, která obsahuje definici názvu daného sloupce. Kromě toho třída také obsahuje odkaz na příslušné požadované nastavení pomocí třídy ColumnGen.

4.2.2 Vytvoření seznamu obsahující pořadí vkládání

Seznam obsahující pořadí pro vkládání slouží pro určení pořadí vkládání jednotlivých tabulek do databázového systému. Díky správnému návrhu databáze a jejich integritních omezení není možné vložit do databáze záznam obsahující referenční klíč na jiný záznam databázové tabulky, který ještě není vytvořen. Pokud bychom se snažili vložit takový záznam, tak by databázový server tuto operaci odmítnul, z toho důvodu je potřeba zaručit, že tabulky budou ukládat do databázového serveru ve správním pořadí. Seznam vkládání se vytvoří za pomocí stromu fyzického návrhu, algoritmus použije metodu prohledávání do hloubky a postupně si ukládá identifikátory tabulek do dvourozměrného pole. První rozměr udává aktuální hloubku stromu a druhý rozměr určuje pořadí vkládání pro danou hloubku stromu. Vkládání na určité hloubce stromu může být zaměnitelné.



Obrázek 11: Strom fyzického návrhu

1		
3	2	
4	5	6
8	9	

Obrázek 12: Matice pořadí vkládání dat do databázového schématu

Po ukončení prohledávání stromu do hloubky se dvourozměrné pole převede na seznam, a to takovým způsobem, že se projde od hloubky s nejvyšším indexem až k nejmenšímu a postupně se vkládá do jednorozměrného pole.

4.2.3 Vytvoření seznamu obsahující pořadí generování

Pro úspěšný proces generování dat je třeba zajistit, aby se jednotlivé sloupce tabulek vytvářely ve správném pořadí. Pokud bychom se například snažili vytvořit nejprve cizí klíče odkazující se na neexistující primární klíče, tak by se program snažil odkázat na neexistující hodnotu a uvízl by v mrtvém bodě.

Prvním krokem pro určení správného pořadí generování je roztřídění všech sloupců jednotlivých tabulek do dvou množin. První množina bude obsahovat sloupce obsahující nějakou závislost na sloupce ostatní a druhá množina bude obsahovat na sobě nezávislé sloupce. Roztřídění se provede na základě informace, zda daný sloupec tabulky obsahuje prerekvizitu generování jiného sloupce tabulky. Pokud sloupec obsahuje takovou prerekvizitu vloží se do množiny číslo jedna, dále se do této množiny vloží i sloupec, na který původní sloupec odkazuje. V praxi to znamená, že všechny sloupce definující primární klíče, nebo cizí klíče budou obsaženy v první množině.

Po roztřídění všech sloupců na závislé a nezávislé množiny sloupců, je potřeba vytvořit pořadí generování. Nezávislá množina obsahující všechny sloupce bez jakýchkoliv vazeb na sloupce jiné, je možná vygenerovat libovolně zvoleným pořadí generování. Pořadí generování se provede klasickým způsobem a to průchodem všech prvků první množiny od nultého indexu až po index nejvyšší. Určení pořadí generování u množiny obsahující závislé sloupce již není tak jednoduché, z daného seznamu závislých sloupců je potřeba vytvořit pořadí generování takovým způsobem, aby nebyly porušeny jednotlivé odkazy na požadované sloupce.

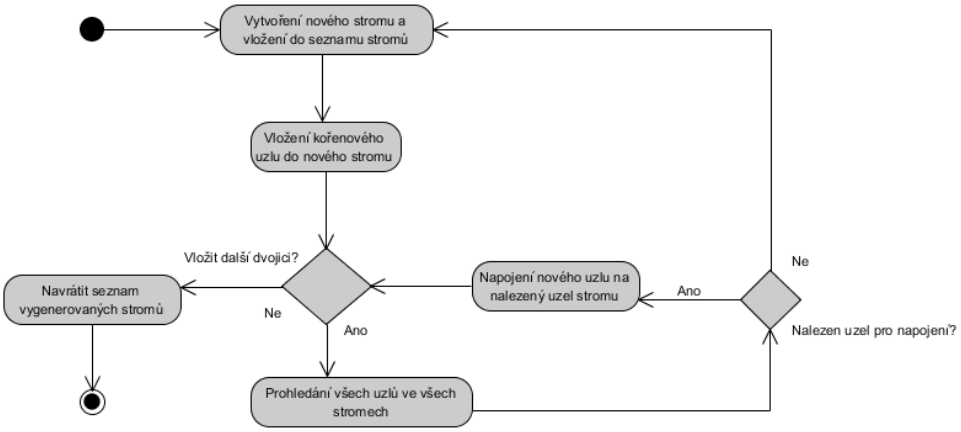
Množina dva obsahuje dvojice na sobě závislých sloupců tabulky, tyto dvojice se postupně spojí do stromu obsahující pořadí generování takovým způsobem, že se nejprve vyhledá, zda daný uzel pro napojení existuje, pokud uzel existuje, připojí se závislá dvojice na daný uzel, pokud ne, tak se daná dvojice nepřipojí, ale přidá se do seznamu pro další prohledávání. Pokud hledaný uzel není nalezen v daném stromu generování, tak je vytvořen nový strom a daná závislost je vložena do nově vytvořeného stromu. Stromy se následně procházejí od listů směrem ke kořenům a jednotlivé úrovně se generují nezávisle na sobě. Samotný strom generování se skládá z uzlů objektů třídy GenTask.

Funkce pro vytvoření seznamu stromů obsahující pořadí generování se nazývá CreateGenerationListTree a tato funkce na vstupu přijímá seznam definic uzlů sloupce v podobě třídy ColumnGen. Funkce projde všechny definice a poskládá závislé uzly do stromových struktur obsahující pořadí generování.

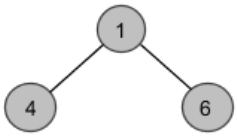
Z příkladu stromu číslo 1 se vytvoří skupiny pro generování v následujícím pořadí: *[4, 6]*, *[1]* a z příkladu číslo 2 potom *[5, 3]*,*[7]* a *[2]*.

4.3 Nastavení generování

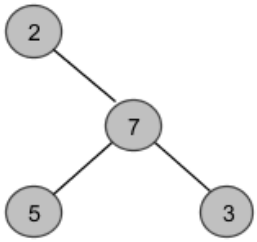
Další fázi je potřeba nastavit požadované vlastnosti generování na základě získaných dat o fyzickém návrhu databáze. Nastavené generování probíhá částečně automaticky a částečně ma-



Obrázek 13: Aktivitní diagram postupu vytvoření seznamu stromů obsahující pořadí generování



Obrázek 14: Příklad stromu určující pořadí generování #1



Obrázek 15: Příklad stromu určující pořadí generování #2

nuálním zásahem uživatele do procesu generování. Po načtení potřebných dat ze serveru aplikace automaticky přednastaví příslušné parametry a následně předá vygenerované nastavení uživateli pomocí uživatelského rozhraní uživateli pro kontrolu vygenerovaného nastavení a dodatečné manuální nastavení.

4.3.1 Automatické nastavení generování

Další fázi je potřeba nastavit požadované vlastnosti generování na základě získaných dat o fyzickém návrhu databáze. Nastavené generování probíhá částečně automaticky a částečně manuálním zásahem uživatele do procesu generování. Po načtení potřebných dat ze serveru aplikace automaticky přednastaví příslušné parametry. Následně předá toto vygenerované nastavení uživateli pro kontrolu zvoleného nastavení a dodatečné manuální nastavení za pomocí uživatelského rozhraní.

Mezi automaticky nastavené parametry a vlastnosti patří vytvoření odpovídajících datových generátorů pro jednotlivé atributy tabulek, nastavení příslušných primárních klíčů, nastavení propojení tabulek pomocí cizích klíčů a identifikace vzorů obsažených v databázovém schématu. Pokud se v návrhu identifikuje vzor, automaticky se aplikuje přislušné nastavení daného vzoru na získáné databázové schéma.

4.3.2 Vzory struktur databázových tabulek a sloupců

V relačních návrzích databází se často opakují stejné nebo podobné uskupení tabulek a sloupců. Takové uskupení, které se nachází v návrhu dostatečně často a obsahuje totožné vlastnosti na vkládána data nazýváme vzorem. Pokud dokážeme takové vzory identifikovat ve fyzickém návrhu databáze, jsme schopni jej využít k vhodné aplikaci netriviálního nastavení generování. Netriviálním nastavením se myslí takové natavení, které se neaplikuje pouze na základě datového typu nebo požadovaného integritního omezení daného sloupce, ale na základě širšího kontextu. Rozlišujeme tři úrovně vzorů: bodový vzor, tabulkový vzor a hierarchický vzor.

4.3.3 Bodový vzor

Bodový vzor je nejsnadnějším vzorem, který generátor dokáže identifikovat v návrhu a aplikovat na něj předem známé nastavení. Takových vzorů většinou je v návrhu nejvíce, jelikož však se jedná o nejjednodušší reprezentaci vzoru je zde největší pravděpodobnost mylného identifikování v návrhu a aplikace špatného nastavení generování. Bodový vzor je definován jako název datového sloupce, datový typ sloupce a požadované nastavení.

Tabulka 4: Příklad bodového vzoru

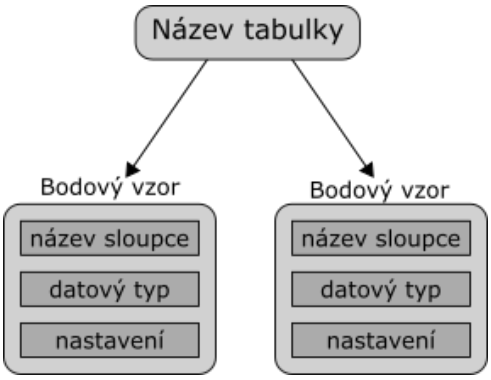
Název	Datový typ	Požadované nastavení
Pojmenování atrubutu tabulky	VARCHAR	ColumnGen

Tabulka popisuje definici bodového vzoru. Kromě identifikace bodový vzor obsahuje již zmíněné požadované nastavení. Toto nastavení je typu třídy ColumnGen, stejně jako je definice tabulky složená z jednotlivých sloupců typů ColumnGen. Díky tomuto faktu je možné do požadovaného nastavení uložit všechna možná potřebná nastavení pro aplikaci na určitý sloupec tabulky. Pokud je bodový vzor identifikován ve fyzickém návrhu databáze, provede se zkopírování veškerého nastavení ColumnGen bodového vzoru do ColumnGen příslušného sloupce tabulky.

4.3.4 Tabulkový vzor

Tento tabulkový vzor se skládá z jednoho nebo více bodových vzorů v rámci jedné společné databázové tabulky. Jedná se o jeden z nejužitěčnějších vzorů. Důvodem je, že je možné upřesnit identifikaci bodového vzoru o rozšíření identifikace pomocí názvu tabulky a upřesnit tak definici bodového vzoru, na který se má aplikovat požadované nastavení. Příkladem může být bodový

vzor se jménem "název". U takové bodového vzoru bez znalosti jména tabulky není jasné, čeho přesně se název týká. U tabulkového vzoru se jménem tabulky "auto"a bodovým vzorem "název"už se dá lépe usuzovat, o jaký název se jedná a jaké nastavení by se mělo aplikovat pro generování dat.



Obrázek 16: Složení tabulkového vzoru

4.3.5 Hiearchický vzor

Pokud vezmeme v úvahu provázání jednotlivých tabulek, je možno ve stromové struktuře reprezentující celý databázový návrh identifikovat hierarchický vzor. Hierarchický vzor se skládá z minimálně dvou tabulkových vzorů a jejich vzájemného provázání.

4.3.6 Hiearchické uskupení vzorů

Graf $A = (VA, EA)$ je podgrafem grafu $B = (VB, EB)$ jestliže platí, že $VA \subseteq VB$ a zároveň $EA \subseteq EB$. Jestliže graf C je podgrafem D a zároveň graf D je podgrafem E potom musí platit, že graf D je také podgrafem E. Pomocí takového to pravidla lze vzory uskupit do hierarchické struktury a prohledávat pouze tak dlouho, dokud byl aktuální vzor nalezen. Algoritmus začne vyhledávat od listů stromu a označí, zda byli jednotlivé vzory nalezeny. Prohledá-li algoritmus všechny vzory v dané úrovni, přesune se o úroveň výše, zde již prohledává pouze ty vzory, u kterých byli nalezená všichni jeho potomci.

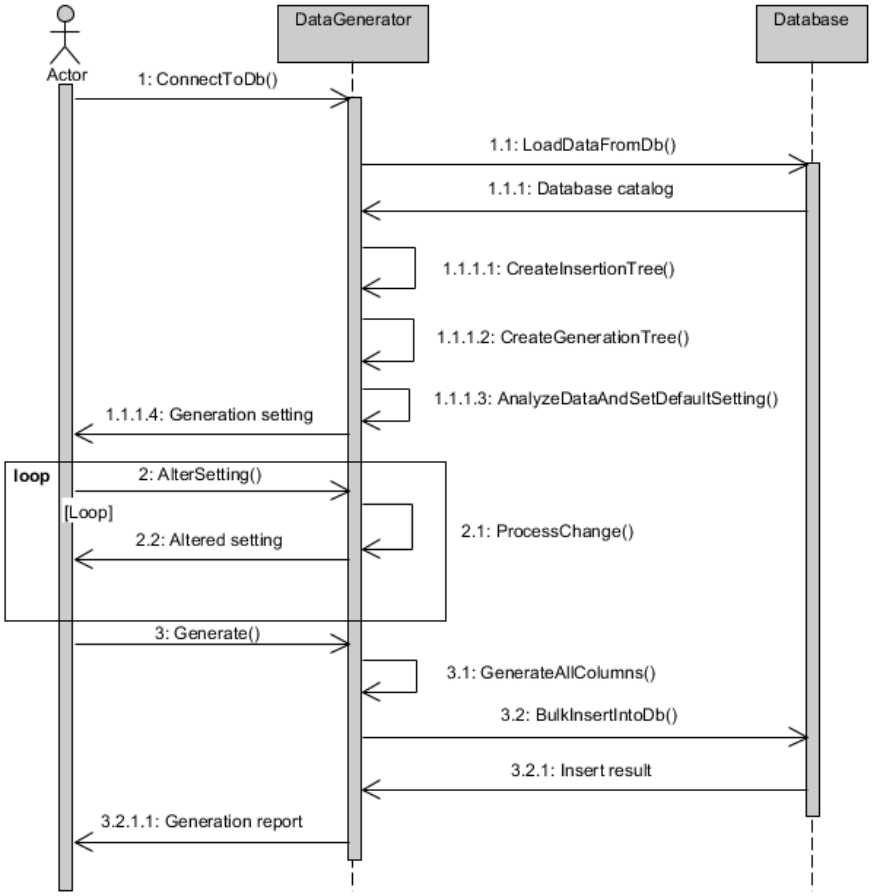
Bodový vzor je podgraf tabulkového vzoru a tabulkový vzor je podgraf hierarchického vzoru. Pro velké množství definovaných vzorů je výhodné jejich vzájemné seskupení do úrovní a v případě porovnávání, vždy začít s menšími podgrafy, tedy bodovými vzory až po celkové grafy tedy hierarchické vzory. Pokud bodový vzor není nalezen ve fyzickém návrhu dané databáze je jasné, že nemůže být nalezen ani tabulkový vzor skládající se z tohoto bodového vzoru. V takovém případě již není třeba porovnávat zbylé vzory v dané hierarchii a algoritmus se může přesunout na další porovnání bodového vzoru.

4.3.7 Manuální nastavení generování

Uživatel pomocí uživatelského rozhraní má možnost dodatečně nastavit generování. Uživatelské rozhraní poskytuje náhled na stromovou strukturu obsahující informace načtené databáze. Tato struktura obsahuje prvky reprezentující databázové tabulky a sloupce jednotlivých tabulek. Pro úpravu nastavení generování u jednotlivých sloupců uživatel vybere sloupec, který si přeje upravit. Po výběru se mu zobrazí příslušné nastavení zvoleného sloupce spolu s vyobrazením možných specifických nastavení. Kromě úpravy nastavení generování sloupců má uživatel možnost také upravit nastavení pro příslušné databázové tabulky.

4.4 Generování dat

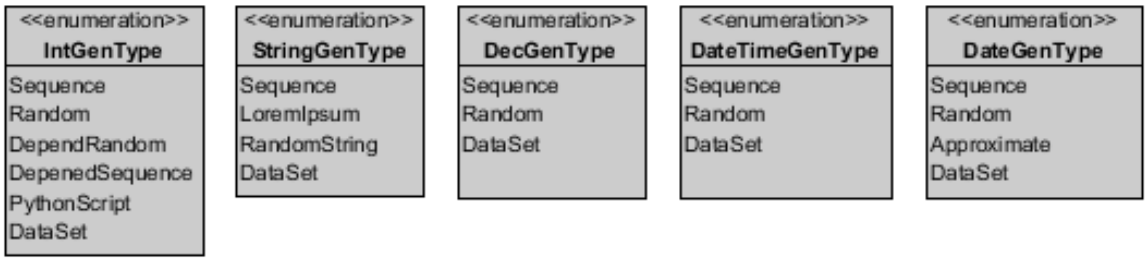
Data jsou generována nejprve sloupec po sloupci a uložena do paměti, následně po vygenerování všech sloupců jsou rekonstruovány jednotlivé paměťové tabulky a uloženy do databázové tabulky.



Obrázek 17: Zjednodušený třídní diagram s postupem přípravy a generování dat

4.4.1 Možnosti generování jednotlivých datových typů

Řešení podporuje generování základních datových typů s velkých počtem možností nastavení generování. Do základní funkce generování datových typů patří generování celých čísel, desetinných čísel, datumů, časů a znakových řetězců. Příslušné nastavení generování jako je generování sekvencí, náhodné generování nebo generování za pomocí vytvořených datových sad a zvoleného distribučního rozdělení se provádí pomocí nastavení vlastností třídy GenType a příslušných parametrů generování. GenType je výčtový datový typ, který je definován rozdílně pro určité datové typy.



Obrázek 18: Definice výčtových typů pro nastavení jednotlivých generátorů datových typů

4.4.2 Datové knihovny

Řešení používá sadu synonym pro určení různých názvů tabulek a sloupců se stejným významem, datové sady obsahující vlasnosti skutečného světa, jako jsou sada křesních jmen, sada názvů států a podobně. Kromě zmíněných dvou obsahuje řešení i sadu vzorů identifikovatelných v databázovém návrhu. Tyto informace se načítají z datových knihoven uložených na disku. V programu je nastavena cesta ke knihovně obsahující data, dále jsou pak již data načítána z pevně dané souborové struktury. Jednotlivá data jsou uložena ve formátu XML a každý XML stromový formát odpovídá reprezentaci dané třídy.

Tabulka 5: Struktura datové knihovny a její popis

Adresář	Typ souboru	Třída	Popis
Synonyms	.gsyn	Synonym	Jenotlivé soubory obsahující seznamy názvů se stejným významem a další potřebná data třídy Synonym.
WordsLibrary	.list	DataSetList	Jednotlivé soubory obsahují seznam s datovými sady skutečného světa a další potřebná data třídy DataSetList
Patterns	.gptr	Pattern	Jednotlivé soubory obsahují kompletní informace pro definici indetifikovatelných vzorů v návrhu databázového schématu

4.4.3 Analýza sady schémat pro nalezení a vytvoření vzorů, synonym

Pro výběr a nastavení vyhledávaných vzorů je potřeba tyto vzory nejprve identifikovat a řádně definovat. Pro identifikaci vzorů jsem využil dostupnou sadu vytvořených databázových schémat studentů, v rámci výuky dabázových systémů na univerzitě. Získaná schémata jsem uložil ve formě tabulky do souboru formátu CSV. Z tohoto souboru jsem dále vycházel během analýzy.

Tabulka 6: Základní údaje analyzovaného souboru

Počet	Popis
292	Celkový počet uživatelů
920	Celkový počet databázových tabulek
2353	Celkový počet atributů databázových tabulek

Prvním krokem analýzy sady schémat je nalezení synonym. Synonyma jsou v tomto smyslu myšlena slova upravená o různé prefixi, nebo postfixy popisující ten samý objekt. Příkladem mohou být "fname", "firstname", "FirstName"všechny tyto názvy popisující stejný objekt a to objekt obsahující hodnoty křestních jmén. Pro získání jednotlivých synonym jsem porovnal každý název sloupce s každým ostatním sloupcem. K porovnání jsem použil rozšířený vzorec výpočtu Levenstainove vzdálenosti o vzatí v úvahu délku porovnávaných slov. Tento vzorec nabývá hodnot od 0 do 1, kde čím se výsledná hodnota blíží více k hodnotě 1, tím více jsou si řetezce podobné. Pomocí rozšířeného vzorce ohodnocení podobnosti jsem dokázal rozlišit podobnost různě dlouhých slov se stejným počtem rozdílných znaků. Po získaní hodnoty určující vzájemnou podobnost slov jsem stanovil hladinu určující minimální nutnou podobnost pro zařazení do jednoho synonyma.

Vzorec pro určení podobnosti P dvou řetezců:

$$P = \frac{L(S_1, S_2)}{N_1 + N_2} \tag{1}$$

L funkce počítající Levenstainovu vzdálenost

S_1 řetezec čislo jedna

S_2 řetezec číslo dva

N_1 počet znaků řetezce číslo jedna

N_2 počet znaků řetezce číslo dva

4.4.4 Popis a implementace Levenstainové vzdálenosti

Někdy nazývána také editační vzdálenost, je řetězcová metrika pro měření editačních rozdílností mezi dvěma řetězci. Pokud máme řetězec A, a řetězec B potom vzdálenost v počtu mazání,

vkládaní nebo substitucí potřebných změn pro transformaci z řetězce A na řetězec B se nazývá Levenstainova vzdálenost.

Používá se pro kontrolu pravopisu, detekci plagiátů nebo pro rozpoznávání řeči. [12]

4.4.5 Generování pomocí datových sad

Pokud existuje požadavek na generování dat takový, že data musí splňovat nějakou vlastnost skutečného světa, není možno vytvořit tyto data uměle. Příkladem může být požadavek na vygenerování křestních jmen, příjmení a nebo názvů států. V takovém případě se musí data vytvořit za pomocí předvytvořené datové sady, kterou řešení načte a použije během generování. Řešení načítá informace o datových sadách z adresáře WordsLibrary datové knihovny.

4.4.6 Vytvoření datových sad

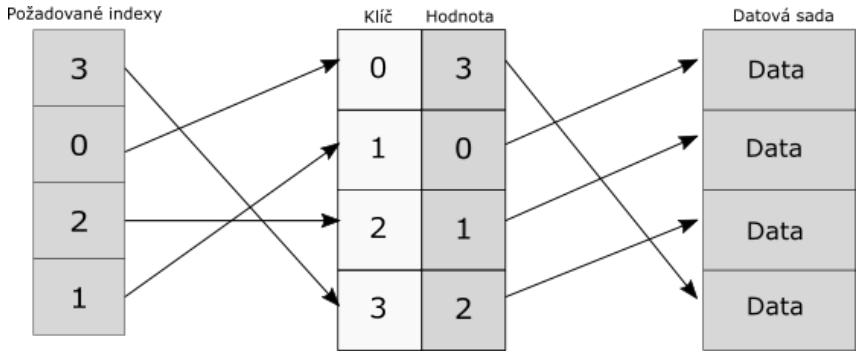
Pro využití datových sad při generování je potřebné tyto datové sady nejprve vytvořit. Prvním krokem při vytváření datových sad je zhodnocení využití jednotlivých datových sad. Nemá cenu vytvářet datové sady, které se ve schématech databází nevyskytují, a tak tyto datové sady nebudou využity. Jakmile jsou zhodnoceny jednotlivé datové sady a odhadnuty četnosti jejich využití zvolil by se výběr nejčastěji používaných datových sad a tyto sady by se vytvořily za pomocí skriptu nebo manuálního sestavení do podporovaného XML formátu knihovny navrženého řešení.

4.4.7 Promíchaní datové sady

Aplikace umožňuje promíchání pořadí datové sady a umožnění tak generování různého vzorku při opakovaném použití stejného nastavení pravděpodobnostního rozdělení a datové sady. K promíchání je využita hashovaná tabulka. Hashovaná tabulka je datová struktura vhodná pro použití při vyhledávání. Hashovací tabulka spojuje hašovací klíče s odpovídajícími hodnotami. Hodnota klíče je vypočítaná na základě hašovací funkce a obsahu položky. V našem případě je použita hashovaná tabulka s přímým hashovaním, kdy hashovací funkce promítá vždy jeden klíč na právě jednu odpovídající hodnotu. V okamžiku požadavku na hodnotu datové sady se nejprve program podívá do hashované tabulky takovým způsobem, že index z požadavku na datovou sadu použije jako klíč pro hashovanou tabulku a získanou hodnotu pak použije jako nový index pro datovou sadu. Pokud hodnota tabulky s daným klíčem je nulová, nechá program vygenerovat unikátní číslo, a to uloží do aktuální hodnoty.

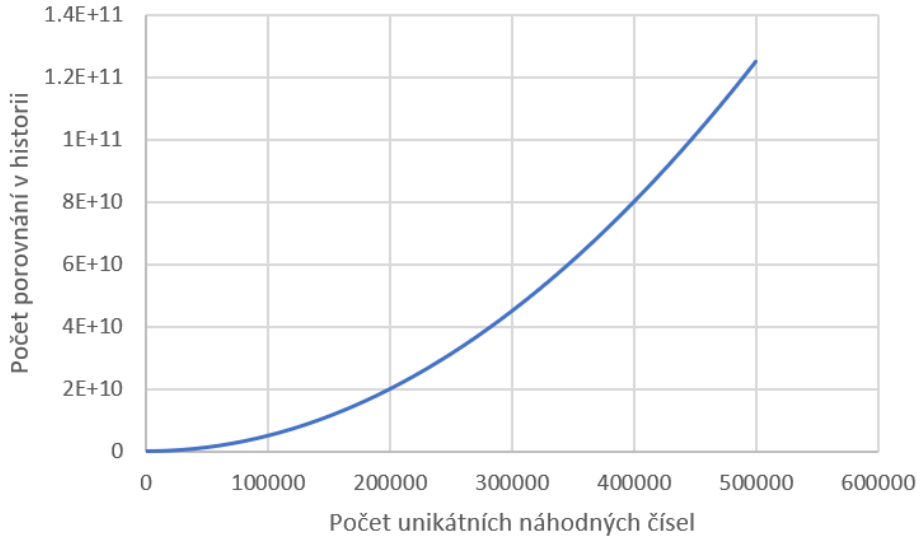
4.4.8 Generování složených klíčů a unikátních náhodných čísel

Standartní přístup generování unikátních náhodných čísel pomocí paměti obsahující historii vygenerovaných čísel a opakovaného volání generování čísel pro již jednout vygenerovaná čísla selhává u požadavku na generování velkého množství dat. Důvodem selhání je rychle se zvětšující počet nutných jednotlivých porovnání v historii se zvyšujícím se počtem požadovaných



Obrázek 19: Promíchání datové sady pomocí hashované tabulky

unikátních čísel. Ještě větší problém nastává při snaze generovat unikátní kombinace čísel pro použití u složených klíčů.



Obrázek 20: Počet porovnání hodnot v historii v závislosti na množství požadovaných hodnot

Proto bylo pro generování velkého množství unikátních čísel zvoleno jiného přístupu, a to vytvoření unikátní posloupnosti s předem určeným konečným počtem čísel. Tato sekvence byla následně promíchána zvoleným promíchávacím algoritmem. Při prvním požadavku na navrácení náhodného čísla se navrátí hodnota na pozici nula a zvětší se čítač určující první nepoužité číslo o jednu pozici. Při každém dalším požadavku na náhodné číslo se navrátí hodnota s pozicí uloženou v čítači a čítač se opět zvýší o jednu pozici, takto lze pokračovat až do využití všech předem vygenerovaných náhodných čísel.

4.4.9 Prohazovací algoritmy

Prohazovací algoritmy slouží k promíchaní pořadí pole o předem známe velikosti. Prohazovací algoritmus je pak možno použít pro generování unikátních náhodných čísel, a to takovým způ-

sobem, že pole naplněné hodnotami 1 až N promícháme, a následně pak vracíme promíchané hodnoty od prvního indexu až po poslední při každém zavolání požadavku na náhodné číslo.

4.4.10 Fisher–Yates shuffle

Fisher-Yates prohazovací algoritmus funguje na principu opakovaného náhodného prohození dvou prvků v jednorozměrném poli prvků. Máme-li pole prvků o velikosti N potom algoritmus provede prohození právě N krát, z toho vyplývá že asymptotická náročnost algoritmu se rovná O(N).

4.4.11 Náhodný generátor bitů

Náhodný generátor bitů zkráceně RBG z anglického překladu „random bit generator“ je zařízení nebo algoritmus, který dokáže vygenerovat posloupnost statisticky nezávislých a nezaujatých binárních číslic. Náhodný generátor bitů lze použit například k vygenerování čísla v rozsahu $[0, n]$ takovým způsobem, že vygenerujeme posloupnost bitů dlouhou $[\ln n] + 1$ a převedeme tuto posloupnost na číslo. V případě, že výsledek překročí n, můžeme jej zahodit a vygenerovat novou sekvenci. [2]

4.4.12 Pseudonáhodný generátor bitů

Ve skutečných podmínkách není jednoduché navrhnout generátor skutečných náhodných bitů, a proto se často používá pseudonáhodný generátor bitů. Pseudonáhodný generátor bitů zkráceně PRBG z anglického překladu „pseudorandom bit generator“ je deterministický algoritmus, který k dané skutečné náhodné binární posloupnosti délky k, vrací binární posloupnost délky $l >> k$, která se tváří jako nahodilá. Vstup do pseudonáhodného generátoru bitů se nazývá seed a výstup se nazývá pseudonáhodná sekvence bitů. [2]

4.4.13 Hardwarově založené generátory

Jsou založeny na odhalení nahodilosti objevující se v některých fyzikálních jevech.

Příklady takových jevů mohou být:

1. Uplynulý čas do doby vyzáření částice během radioaktivního rozpadu.
2. Termální šum z polovodičové diody nebo rezistoru.
3. Frekvenční nestálost volně oscilujícího oscilátoru.
4. Množství nabití kovového izolátoru polovodičového kondenzátoru během pevné časové periody.

[2]

Proto, aby bylo zaručeno, že nebude manipulováno s generátory, musely by první dva typy být externě odděleny od jednotky využívající generátory. Zbylé dva přístupy by mohly být integrovány do zařízení pomocí teplotně-nezávislého hardwaru tak, aby byly odstíněny od případné manipulace. [2]

4.4.14 Softwarově založené generátory

Návrh softwarově založeného generátoru bez možnosti využití nahodilostí vyskytovaných ve fyzikálních jevech je poměrně náročné. Při návrhu se musí dbát na fakt, aby uživatel nebyl schopen manipulovat s generátorem pseudonáhodný dat, a aby generátor využíval co možná nejvíce dostupných systémových možností pro generování dat.

Softwarově založené generátory dat nejčastěji využívají zdrojů systému jako jsou:

1. systémový takt
2. uplynulá doba mezi stisky klávesy či pohyby myši
3. obsah vstup/výstupních vyrovnávacích pamětí
4. hodnoty operačního systému jako je zátěž systému či statistika sítě

[2]

4.4.15 Lineární kongruentní generátor

Lineární kongruentní generátor zkráceně LKG, je jeden z nejstarších a nejpoužívanějších generátorů pseudonáhodných čísel. Jeho široké rozšíření bylo umožněno hlavně díky jednoduché implementaci a na dobu poměrně dostatečný výkon generovaných pseudonáhodných posloupností. Generování pseudonáhodných čísel je definováno na základě rovnice:

$$x_{i+1} = (ax_i + c) \quad mod \quad m \tag{2}$$

[3]

Je taková kde:

x_{i+1} nově vygenerované číslo

a násobitel v podobě předem zvolené konstanty

c inkrement v podobě předem zvolené konstanty

m dělitel s celočíselným zbytkem po dělení

x_i poslední vygenerované číslo

[3]

4.4.16 Více vláknový generátor náhodných čísel

Standartní implementace generátoru náhodného čísla nepočítá s použitím v multi-vláknových aplikací. Pokud tato implementace je bez úpravy použita ve více vláknových aplikací, může dojít ke kolizi při zápisu do paměti, poškození vnitřní struktury generátoru a k narušení funkci generování pseudonáhodných dat.

Aby nedošlo k poškození generátoru dat vlivem souběhu, existují dvě možné implementace. První implementace používá jeden globální generátor dat pro všechny vlákna a vhodně uzamyká přístup k zápisu a čtení pro jednotlivá vlákna. Takový přístup degraduje výkon aplikace, jelikož uzamknutí generátoru zablokuje přístup pro ostatní vlákna, dokud se daný zámek neuvolní a tvoří tak úzké hrdlo výkonu. Druhou variantou je použit pro každé vlákno jeden generátor dat a daná vlákna se potom nebudou dělit o jeden globální generátor dat.

Pokud ve více vláknových aplikacích inicializujeme proměnné třídy Random v téměř stejný okamžik, dostaneme pro první vygenerování čísel u všech vláken stejné počáteční pseudonáhodné číslo. Pokud budou intervaly generování dalších náhodných čísel stejné u všech vláken, potom budou i následná vygenerovaná čísla stejná jako v ostatních vláknech. Takové chování je velice nežádoucí jak u generátoru náhodných čísel, tak i u generátoru pseudonáhodných čísel. Inicializace generátoru probíhá za pomocí tak zvaného seedu. Seed je hodnota, která se nastaví do vnitřní paměti generátoru a z této hodnoty se následně odvíjí další vygenerovaná čísla. Při inicializaci se běžně používá časové razítko, které se převede na číslo a použije jako seed. Pokud tedy všechna vlákna použijí totožné časové razítko je jasné, že vygenerované hodnoty budou také totožné. Pro odstranění takového nežádoucího jevu je zapotřebí inicializovat generátory různou hodnotou, a to nejlépe náhodně vygenerovanou.

V řešení byl implementován globální generátor pomocí návrhového vzoru Singleton [13], který slouží pro inicializaci ostatních generátoru pseudonáhodných čísel v jednotlivých vláknech programu.

4.5 Skriptovací rozhraní

Aplikace poskytuje uživateli širokou škálu možností nastavení generování, od generování základních datových typů spolu s různými možnostmi nastavení až po generování z připravených datových sad a volby distribučního rozdělení. I přes velký počet možností nastavení generování není možno pokrýt všechny možné uživatelské požadavky na generování, a proto aplikace poskytuje možnost dodatečného nastavení generování pomocí skriptovacího jazyka. Pro možnosti skriptování byl zvolen programovací jazyk Python s implementací kompilátoru IronPython.

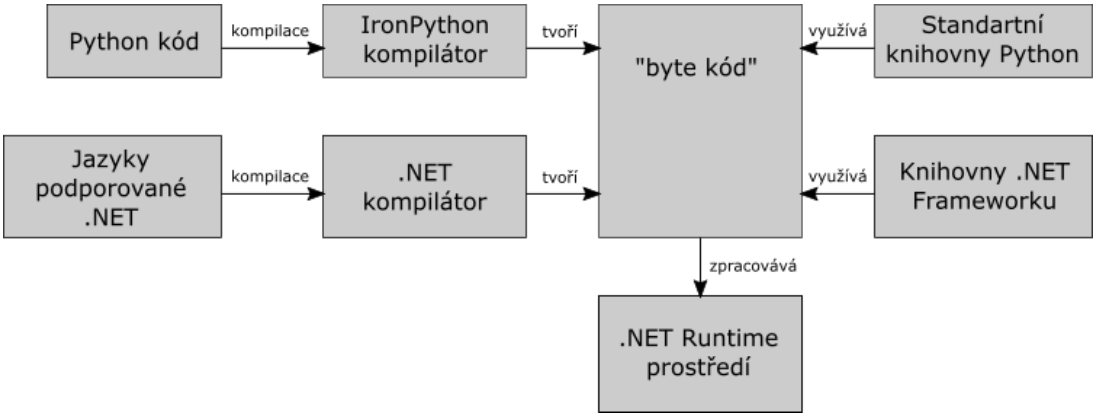
4.5.1 CLR, DLR a IronPython

Framework .NET poskytuje jazykovou interoperabilitu napříč programovacími jazyky. Programy vytvořené .NET Frameworkem se vykonávají v softwarovém prostředí nazývané Common Language Runtime. CLR je aplikace virtuální mašiny, která poskytuje zabezpečení, správu paměti

a správu vyjímek. Virtuální mašina zpracovává předkompilovaný kód nazývaný „byte code“, takovým způsobem že ho za běhu převádí do strojového kódu dané platformy.

Pro potřeby služeb dynamického programování byl na CLR vystavěn systém s názvem Dynamic Language Runtime. Tento systém slouží k implementaci dynamických programovacích jazyků na platformě .NET. Mezi podporované dynamické jazyky patří například Lisp, Smalltalk, JavaScript, PHP, Ruby nebo Python.

IronPython je implementací kompilátoru v jazyce C# pro programovací jazyk Python. IronPython kompiluje program napsaný v Pythonu do „byte kódu“ který je následně zpracován JIT na místo standartního interpreta CPython napsaného v jazyce C, který interpretuje program napřímo. [10]



Obrázek 21: Integrace IronPythonu do architektury .NET frameworku

Rozhrání zpřístupňuje veškerou funkcionalitu poskytovanou standartními knihovnami jazyku Python. Uživatel má také možnosti využít zpřístupněného rozhraní aplikace pro účely naprogramování požadované funkcionality.

Tabulka 7: Zpřístupněná funkcionalita skriptovacího rozhraní

Syntaxe	Popis
<název sloupce>	Symbolický odkaz na instanci třídy typu ColumnGen, obsahující v atributu Name příslušný název sloupce.
<název sloupce>.GenerationCount	Atribut třídy ColumnGen obsahující definovaný počet požadovaného množství generování.
<název sloupce>.Count	Atribut třídy ColumnGen obsahující aktuální počet vygenerovaného množství dat.

Jako výstupní bod Python skriptu je funkce GetString, která musí vracet znakový řetězec. Tento řetězec je následně vkládán na místo sloupce tabulky při generování datové sady. Pro vložení a základní odladění skriptu poskytuje aplikace textové rozhraní se základními prvky validace. Textové rozhraní poskytuje dvě základní funkce, první funkce slouží pro kompilaci

skriptu a druhá je validační funkce. Funkce pro kompilaci poskytuje uživateli informaci o správnosti použité syntaxe ve zdrojovém kódu skriptu. Validační funkce poskytuje kontrolu sématické správnosti. Tato funkce v prvním kroku vygeneruje veškeré závislosti pro daný kontext skriptu. Mezi tyto závislosti patří veškerá data seskupená do dané tabulky, jako jsou generovaná data, nastavené vlastnosti generovaných dat, anebo počet vygenerovaných dat. Následně se pokusí vygenerovat data pro daný sloupec pomocí zkompilovaného skriptu. Skript může být dvojího typu. Prvním typem skriptu je skript neobsahující závislosti na ostatní data dané tabulky a využívající pouze funkcionality Python knihovny. Druhou možností je typ skriptu využívající jak možnosti Python knihovny, tak zpřístupněného rozhraní generátoru a závislosti na ostatní data dané tabulky. Při použití druhého typu skriptu je potřeba dodržet správného pořadí generování dat. Řešení je schopno rozpoznat správné pořadí generování z informací obsažených v databázovém serveru o příslušném databázovém návrhu, pokud však uživatel přidá závislosti na jiných datech dané tabulky, v takovém případě je potřeba manuálně upravit pořadí generování za pomocí elementů uživatelského rozhraní určující pořadí generování.

Řešení obsahuje dva seznamy, které dohromady určují pořadí generování. První seznam obsahuje list generování sloupců dat, které jsou na sobě nezávislé, a tak není potřeba nijak dodržet pořadí generování. Druhý seznam obsahuje list stromů s pořadím generování. Každý strom pořadí obsahuje závislosti jednotlivých dat a provádí generování od potomků směrem ke kořenu stromu. Při definování skriptu pro generování s odkazem na ostatní data tabulky, je vhodným přístupem ošetřit možnost nedostupnosti závislých dat. Tato nedostupnost může vzniknout nesprávným pořadím generování zapříčiněným špatným manuálním nastavením pořadí generování.

```
import sys
def GetString():
      if (surename.Count) >= (firstname.Count):
           return str(surename.Data[(firstname.Count -1)]) + "@mail.com"
     else:
           return "missing data"
```

Výpis 1: Program demonstrující využití reference k vytvoření emailové adresy v jazyce Python

Pokud při vykonání daného skriptu dojde k načítaní neexistujících dat, pak se vloží do výstupních dat hláška definována uživatelem signalizující nedostupnost dat. Pokud by skript neobsahovat kontrolu dostupnosti dat, poté by se generování dat zastavilo s chybovou hláškou na neexistující datový typ.

5 Desktopová aplikace generování dat

Pro účely snadného upravení požadovaného nastavení a vizualizace aktuálního nastaveného generování, poskytuje řešení aplikaci pracující s knihovnou generování dat. Aplikace je navržená tak, aby uživatel s minimálním počtem zásahů do procesu generování byl schopen vygenerovat data, která budou splňovat integritní omezení cílové databáze zvolené pro umístění dat. Pokud uživatel bude mít další požadavky na data kromě splnění integritních omezení databáze, může tyto požadavky nastavit v jednotlivých objektech schématu generování, nebo využít zpřístupněného skriptovacího rozhraní. Aplikace obsahuje sadu pohledů a oken, které jsou navrženy pomocí návrhového vzoru MVVM a vhodně zakomponovány do prezentační vrstvy. Desktopová aplikace je implementována pomocí .NET frameworku a technologie WPF zkratky z anglického názvu Windows Presentation Form. Aplikace využívá funkce knihovny pro generování dat a umožnuje uživateli zasahovat do procesu generování dat, ať už nastavení generování nebo změnou cíle uložení dat. Při návrhu byl kladen důraz na vlastnost nízkého provázání mezi jednotlivými vrstvami softwaru. Pomocí oddělení doménové logiky od prezentační vrstvy bylo dosaženo čistého a přehledného zdrojového kódu. Tento zdrojový kód bude pomocí správného návrhu dále poměrně jednoduše rozšířitelný.

5.1 Nízká úroveň provázání pomocí návrhového vzoru MVVM

Pří návrhu softwarového díla, vzniká častý problém příliš vysoké úrovně provázání mezi prezentační vrstvou a vrstvou doménového modelu. V praxi se tento problém často projevuje potřebným velkým počtem zásahů do prezentační vrstvy při změně v doménovém modelu a naopak. Pro dosažení nízké úrovně provázaní mezi doménovým modelem a prezentační vrstvou byl kladen důraz na využití návrhového vzoru Model-View-ViewModel při návrhu řešení. Samotné řešení je pak rozděleno do tří oddělených modulů s názvy RealDataGenLib obsahující funkce generování dat a veškerou doménovou logiku, RealDataGenGUI obsahující uživatelské rozhraní a RealDataGenTest obsahující sadu funkcí pro testování ověření správné funkcionality.

Návrhový vzor model view viewmodel je vzor oddělující uživatelské rozhraní od business logiky aplikace. MVVM vzor patří do skupiny vzorů nazývané oddělená prezentace. Tato skupina vzorů poskytuje přehledné oddělení uživatelského rozhraní od zbytku aplikace. Pomocí návrhového vzoru MVVM je umožněno efektivnější testování a také snadňější vývoj pomocí odstranění úzké vazby mezi uživatelským rozhraním a bussiness logice. [11]

Pro ulehčení realizace návrhového vzoru MVVM je možno využít frameworku již obsahující tento vzor a urychlení tak vývoje.

Nejpoužívanějšími MVVM framworky jsou:

1. MVVM Light toolkit
2. Prism

3. Caliburn

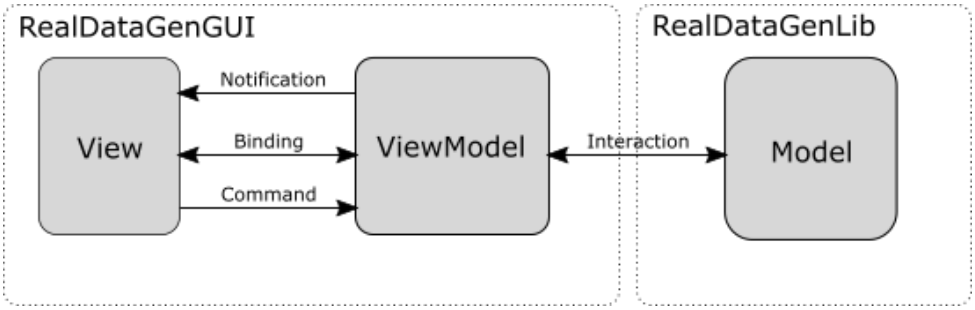
4. ReactiveUI

MVVM frameworky usnadňují vývoj WPF a webových aplikací pomocí vložení vrstvy implementující MVVM vzor mezi prezentační vrstvu a vrstvu modelu. Tato vrstva urychluje vývoj pomocí faktu, že vývojáři nemusí implementovat viewmodel vrstvu pro každý pohled prezentační vrstvy znovu a znovu, ale pouze využijí implementaci frameworku. Jednotlivé frameworky implementují z menší nebo z větší části MVVM vzor. V případě využití pouze malého počtu vlastností ze zvoleného frameworku v aplikaci, může využití obsáhlého frameworku zbytečně nepoměrně rozšiřovat zdrojový kód, celkovou velikost programu a zbytečně zvětšovat náročnost vývoje programu. Jelikož v popsaném řešení bylo využito jen pár základních vlastností MVVM vzoru, tak nebyl při vývoji využit žádný MVVM framework a veškerá funkcionalita byla vytvořena ručně.

Potřebné vlasnosti MVVM vzoru v řešení:

1. Předávání dat mezi modelem a prezentační vrstvou

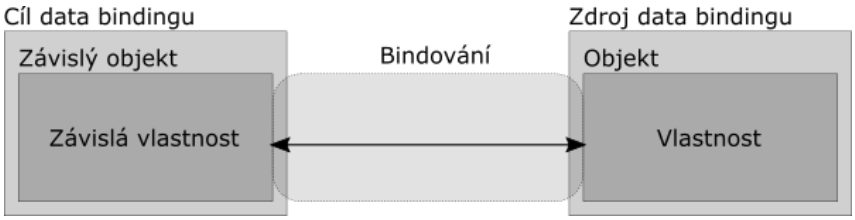
2. Volání funkcí pomocí předávání zpráv/příkazů



Obrázek 22: Návrh řešení za pomocí MVVM návrhového vzoru

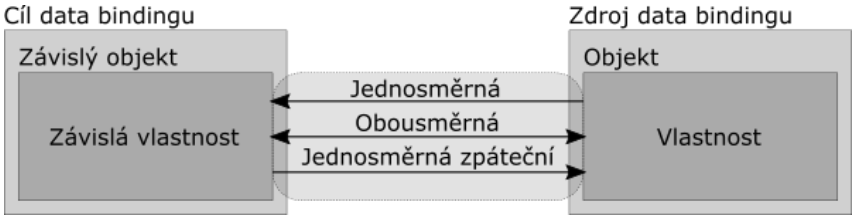
Návrhový vzor MVVM využívá technologie data bindingu a předávání zpráv v podobě příkazů na místo přímé reference. Datové bindování je technika předávání dat ze zdroje do cíle a naopak. Cíl je v našem případě prezentační vrstva a datovým zdrojem je datový model. Pomocí data binding datová vrstva neobsahuje žádné reference na objekty prezentační vrstvy, což snižuje provázání jednotlivých vrstev a usnadňuje vývoj přehledného softwarového díla. Veškerý data binding funguje na základním předpokladu, že v návrhu softwaru máme objekt s vlastnosti, kterou chceme promítnout do vlastnosti jiného cílového objektu s co možná nejnižším stupněm provázání.

Bindování dat může probíhat jednou ze tří možných přístupu přenosu dat. První možností je jednosměrné bindování, kdy se data přesouvají pouze jediným směrem ze zdrojového objektu do cílového objektu. Tato metoda se nejčastěji používá pro needitovatelné elementy uživatelského



Obrázek 23: Data binding [6]

rozhraní jako jsou Labely, nebo vlastnosti daných objektů jako jsou například viditelnost, upravitelnost a podobně. Druhou možností je dvoucestné bindování, kdy se data přenášejí oběma směry, tato metoda se používá pro všechny editovatelné elementy uživatelského rozhraní, jakou jsou TextBoxy, RichTextBoxy, anebo CheckBoxy. Poslední možností je metoda jednosměrného zpátečního přenosu dat z cíle do zdrojového objektu.

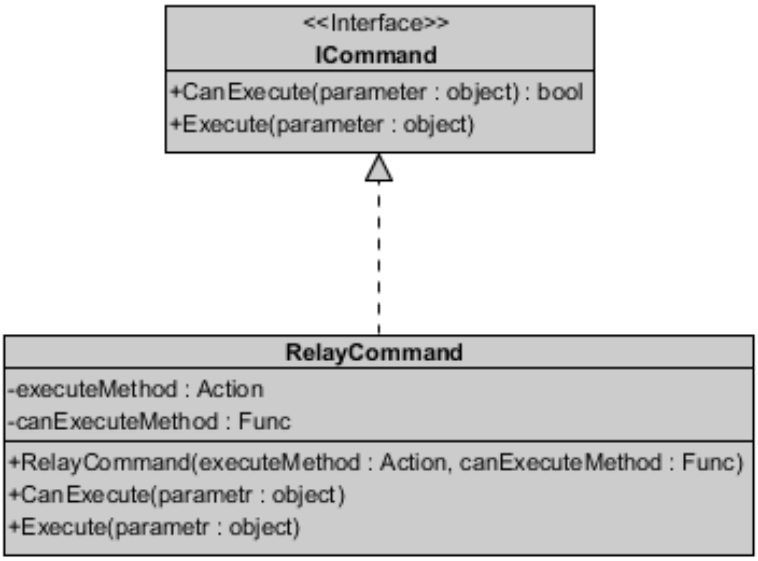


Obrázek 24: Metody bindování [6]

Pro dosažení nízkého provázání u vykonávaní obslužných metod jednotlivých elementů uživatelského rozhraní se využívá předávání zpráv na místo přímé vazby na funkce. Předávané zprávy se také nazývají příkazy z anglického překladu „command“ a fungují na principu zasláním zdroje, v našem případě element uživatelského rozhraní do komunikačního kanálu, kde příslušný příkaz zaregistruje cílová strana v našem případě ViewModel a daný příkaz vykoná. Pomocí nepřímé vazby za pomocí předávání příkazu model ztrácí veškeré závislosti na prezentační vrstvu. Příkaz se skládá z komunikačního kanálu a zprávy obsažené v příkazu. U jednoduchých uživatelských rozhraní, které obsahuji jen malé množství volání funkcí lze definovat pouze jeden příkaz a výběr, která funkce se má ve ViewModelu vykonat, rozhodnout na základě obsahu předávané zprávy. Pokud prezentační vrstva obsahuje velké množství příkazů potřebných k vykonání, jeví se lepší možnosti vytvořit více než jeden komunikační kanál a vhodně seskupit příkazy na základě funkční dekompozice pro daný komunikační kanál.

Pro zajištění komunikace pomocí zasílaní zpráv je potřeba dohody uniformního rozhraní zpráv u jednotlivých stran. Pro tuto dohodu slouží v .NET frameworku rozhraní ICommand, které definuje rozhraní pro zasílání příkazů.

Pro implementaci rozhraní ICommand byla vytvořená třída RelayCommand na základě návrhu vývojářů Karla Shifflett a Joshna Smith [11]. Třída RelayCommand přijímá v konstruktoru odkaz na dvě funkce. Prvním argumentem konstruktoru je predikátová funkce a jako druhý argu-



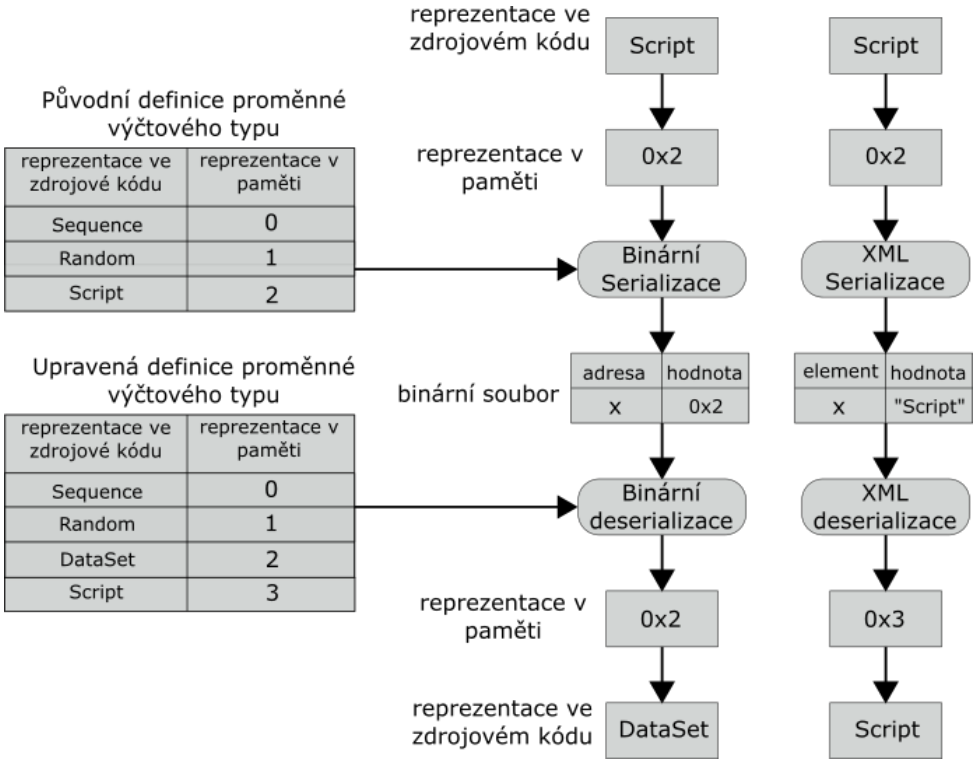
Obrázek 25: Rozhraní pro předávání příkazů mezi objekty

ment funkce vykonávající požadovanou funkcionalitu. Predikátová funkce obsahuje funkcionalitu určující, zda se lze následnou funkcí obsahující požadovanou funkcionalitu vykonat.

5.2 Perzistence a datová serializace

Jako ve většině aplikací i v mém řešení aplikace datového generátoru je potřeba přenést určitá data aplikace z paměti na perzistentní uložiště. Jako perzistentní uložiště byl zvolen pevný disk. Jednotlivá data je pro snadné editování nutné uložit do co možná nejvíce srozumitelné formy, proto se nabízí jako použitelné způsoby metody uložení: ukládání do podoby prostého textu, XML serializace a binární serializace. Pro nestrukturována data byl zvolen způsob uložení prostého textu a pro strukturována data XML serializace. Binární serializace byla zamítnuta z důvodu nekonzistentní reprezentace během vývoje a správy softwaru. Pokud vytvoříme za pomocí binární serializace binární soubor v určité verzi datového modelu a následně se snažíme deserializovat tak původní data s velkou pravděpodobností nebudou shodná s načtenými daty již při drobném zásahu do datového modelu. Příkladem může být i tak miniaturní zásah, jako je prohození nebo promíchání z důvodu rozšíření definice výčtové hodnoty enum.

Ve zdrojovém kódu se u výčtových hodnot enum využívají názvy symbolizující jednotlivé číselné hodnoty výčtového typu. V paměti programu se používají již pouze číselné hodnoty reprezentujícího jednotlivé hodnoty výčtového typu. Pokud tedy provedeme uložení do binárního souboru za použití původní definice výčtového typu a následně provedeme načtení binárního souboru za pomocí definice s prohozenými či rozšířenými výčtovými typy, tak se původní uložená hodnota nebude rovnat nové načtené hodnotě.



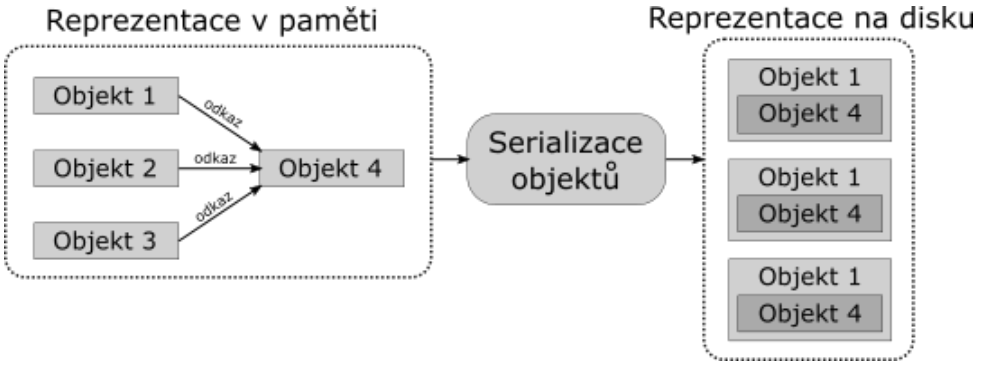
Obrázek 26: Serializace výčtového typu

Popsaný problém se týká pouze binární serializace. Při uložení za pomocí XML serializace se jednotlivé hodnoty výčtového typu uloží vždy ve formě znakového řetězce. Pokud se pokusíme obdobně načíst data jako to bylo v binární serializaci za pomocí upravené definice výčtového typu, tak se původní uložená hodnota a nová načtená hodnota vždy budou rovnat.

5.3 Serializace složených objektů a snížení úrovně provázanosti mezi objekty

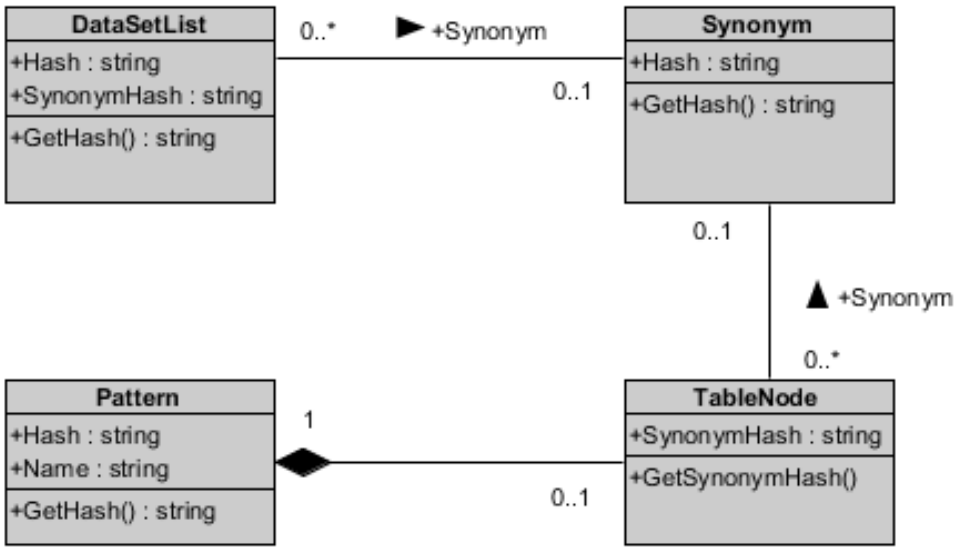
Je běžných případem že se jednotlivé objekty skládají z množiny objektů nižších vrstev. Objekty, ze kterých se složí celkový objekt, nemusí být vždy disjunktní. Pokud se pokusíme takový objekt serializovat vznikne nám v souboru složitá struktura popisující všechny informace daného objektu, pokud budeme serializovat více objektů složených ze společných fragmentů, budou soubory obsahovat ty samé informace několikrát.

K vyřešení tohoto problému je třeba snížit úroveň provázanosti jednotlivých objektů bez jakékoliv ztráty dat. Toho docílíme definováním, která data se mají serializovat a která nikoli. Data primitivních datových typů a disjunktní data budeme ukládat všechny, naopak konjunktivní data budeme při ukládání daného objektu ignorovat. Jelikož je třeba, aby byl objekt možno zpětně rekonstruovat, nahradíme všechny ignorované reference hodnotou identifikátoru na tuto referenci. Jako nejméně vhodný identifikátor se jeví unikátní číslo, důvodem je jeho snadná zaměnitelnost při zpětném dohledávání reference s jiným objektem. Vhodnějším se jeví řetězec popisující danou referenci například „sada_jmen“, nejvhodnějším identifikátorem se pak jeví



Obrázek 27: Serializace složeného objektu

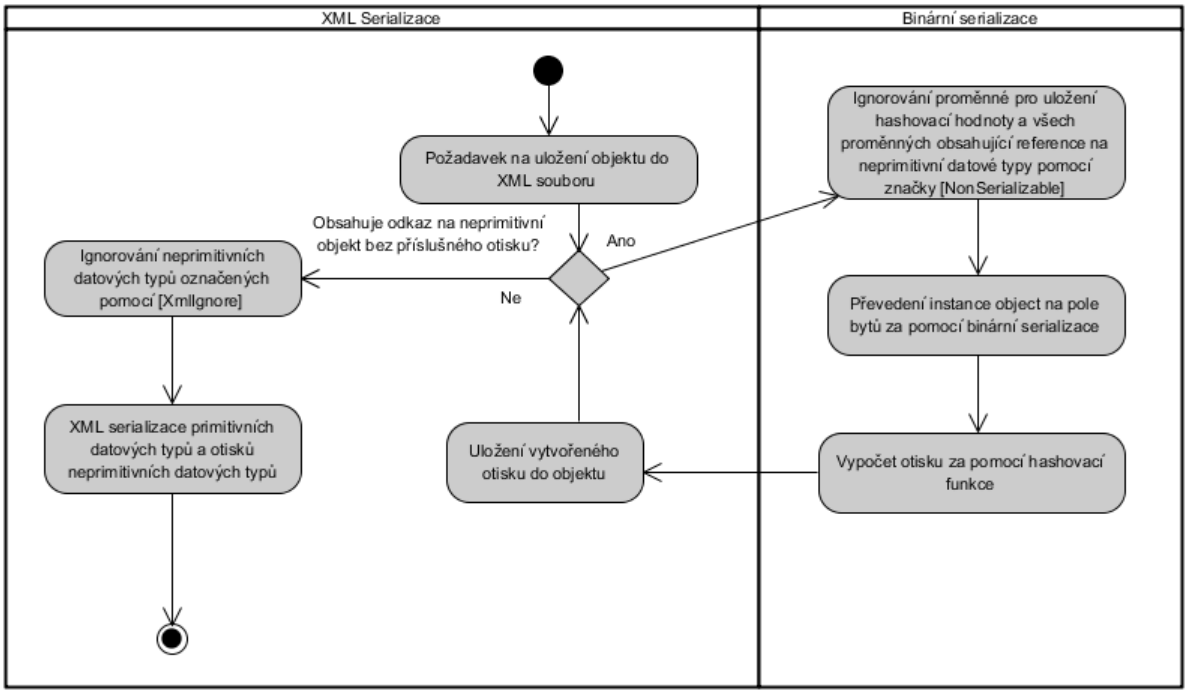
hashovací funkce, která pro vstup používá data odkazovaného objektu a z těchto dat vytváří jedinečný identifikátor.



Obrázek 28: Třídní diagram znázorňující propojení pomocí hashovacího identifikátoru

Hashovací funkce je algoritmus nebo matematický aparát, který pro libovolně dlouhou sekvenci čísel vyprodukuje vždy číslo stejné délky. Toto vyprodukované číslo se nazývá haš, otisk či fingerprint. V případě použití hashovací funkce pro určení odkazovaného objektu, získáváme kromě identifikace jednotlivých odkazů, také pomocí závislosti výstupu z hashovací funkce na vstupních hodnotách i informaci o obsahu objektu. Ve většině případů, pokud dvě výstupní hodnoty hashovací funkce A a B jsou stejné, potom lze předpokládat, že i vstupní hodnoty pro A a pro B byly totožné, toto platí z definice hashovací funkce. Výběr hashovacího algoritmu probíhá na základě požadavku minimalizace možnosti kolize. Kolizí u hashovací funkce se myslí vyprodukování stejné výstupní hodnoty hashovací funkce pro různé vstupní hodnoty. Pravděpodobnost

kolize u hashovací funkce je malá nicméně reálná.



Obrázek 29: Kombinace binární a XML serializace pro uložení objektu s referencemi

Jelikož výpočet hashovací funkce pracuje pouze s číselnými hodnotami, je jako první zapotřebí převést instanci na pole bytů. K tomuto převodu se využívá metoda ObjectToByteArray přijímající jako parametr hodnotu typu object a vracející návratovou hodnotu pole bytů. Funkce využívá k převodu binární serializaci a to takovým způsobem, že serializuje instanci do proudu v paměti. Po navracení vytvořeného pole bytů se následně předá toto pole do hashovací funkce. Jako hashovací funkce byla zvolena metoda hashovaní s názvem SHA256 z anglického názvu „Secure Hash Algorithms“ a to z důvodu kompromisu mezi výkonem a pravděpodobností kolize hashovaných hodnot. Funkce SHA je symetrická bloková šifra, která ve verzi 256 produkuje pro libovolný počet vstupních hodnot otisk pevné délky 32 bytů. Po úspěšném vykonání hashovací funkce se otisk uloží do proměnné, která byla použita na vstupu, aby nedošlo k vytvoření rozdílného otisku při opakování generování, je tato proměnná označena značkou pro ignorování při binární serializaci.

6 Testování a verifikace

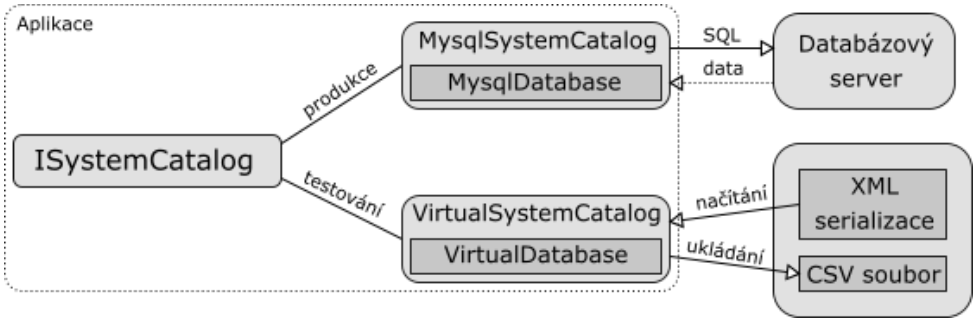
Řešení obsahuje sadu unit testů pro testování klíčové funkcionality datového generátoru. Do skupiny klíčové funkcionality jsou zařazeny metody pro perzistentní ukládání a načítání dat, funkce generování hodnot s určenými distribučními rozloženími, metody porovnávání stromů obsahující pořadí generování, metody načítaní slovníků a datových sad potřebných pro generování dat.

Tabulka 8: Seznam použitých unit testů

Název unit testu
DataSetLibraryLoad
CompleteTreeStructureComparison
FindTreeInTreeComplexTest
FindTreeInTreeSimpleTest
FullDataSerializationTest
GenerationTreeTest
LibraryLoadTest
PatternSerializationTest
QuantumRandomTest
RandomOrgBasicTest
TestExponentialDistributionImage
TestGaussDistributionImage
TestHashingFunction
TestPythonScripting
TestSynonymLibSerialization
TestUniformDistributionImage
TreeStructureComparison
XmlSerializationTest

Při testování aplikace je z důvodu načítání a ukládání dat potřebný neustálý přístup k databázovému serveru, tato potřeba může být v případech nedostupnosti lokálního databázového serveru na testovací mašině nebo v případě nedostupnosti sítě ke vzdálenému databázovému serveru omezující, proto byla vytvořena pro účely testování vrstva softwaru simulující funkcionalitu databázového serveru. Tato vrstva umožňuje načítat předem vytvořená požadovaná data z lokálního perzistentního uložiště. Jelikož tento virtuální databázový server realizuje rozhraní přístupu k databázovému serveru, lze jednoduše měnit použité databázové servery úpravou na jediném místě ve zdrojovém kódu.

Pro potřeby uložení vygenerovaných dat virtuální databázový server poskytuje možnost uložení dat textového dokumentu splňující požadavky formátovaní CSV souboru. Rozlišení, zda se použije simulovaný databázový systém, nebo skutečná implementace databázového systému se provádí na základě podmíněného větvení programu preprocesoru.



Obrázek 30: Testování pomocí simulovaného databázového systému

6.1 Pravděpodobnostní rozdělení

Pravděpodobnostní funkce se používá k popisu rozdělení diskrétní náhodné veličiny. Rozdělení pravděnepodobnosti náhodné veličiny je pravidlo, kdy každému jevu přidělíme určitou pravděpodobnost, se kterou daný jev nastane.

6.2 Distribuční funkce

"Nechť X je náhodná veličina. Reálnou funkci $F(x)$ definovanou pro všechna reálná x vztahem"[4]

$$F(x) = P(X < x)$$

Obrázek 31: Definice distribuční funkce[4]

"nazýváme distribuční funkcí náhodné veličiny X."[4]

"Distribuční funkce je tedy funkce, která každému reálnému číslu přiřazuje pravděpodobnost, že náhodná veličina nabude hodnoty menší než toto reálné číslo."[4]

6.3 Rozdělení pravděpodobnosti spojité náhodné veličiny

Spojitá náhodná veličina je taková veličina, jejíž distribuční funkce je také spojitá.

6.4 Hustota pravděpodobnosti

Hustota pravděpodobnosti se používá k popisu rozdělení spojité náhodné veličiny. "Hustota pravděpodobnosti $f(x)$ spojité náhodné veličiny je reálná nezáporná funkce taková, že"[4]

$$F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t \text{ pro } -\infty < x < \infty.$$

Obrázek 32: Definice hustoty pravděpodobnosti [4]

6.5 Rovnoměrné rozdělení

„Jde o rozdělení, jehož hustota pravděpodobnosti je konstantní na nějakém intervalu (a; b) a všude jinde je nulová.“ [4] Standartní implementace generování pseudonáhodných čísel generují náhodné veličiny právě tohoto distribučního rozdělení. Pro umožnění využití definuje knihovna třídu UniformDistribution, která zaobaluje základní implementaci o možnost použití ve více vláknových aplikacích.

X ... náhodná veličina s rovnoměrným rozdělením na intervalu (a; b)

$$X \to R(a; b)$$

[4]

Hustota pravděpodobnosti:

$$f(x) = \begin{cases} \frac{1}{b-a} & x \in (a; b) \\ 0 & x \notin (a; b) \end{cases}$$

Obrázek 33: Hustota pravděpodobnosti rovnoměrného rozdělení [4]

Distribuční funkce:

$$F(x) = \begin{cases} 0 & x \in (-\infty; a\rangle \\ \frac{x-a}{b-a}, & x \in (a; b) \\ 1 & x \in \langle b; \infty) \end{cases}$$

Obrázek 34: Distribuční funkce rovnoměrného rozdělení [4]

6.6 Normální rozdělení

Mezi jedno z nejdůležitějších rozdělení vyskytujících v přírodních jevech, nebo technických jevech patří normální rozdělení. Normální rozdělení také nazývané Gaussovo rozdělení, nebo obecné normální rozdělení se sice přesně řídí pouze jen v malém počtu náhodných veličin, ale jeho význam spočívá ve faktu, že toto rozdělení v některých případech přijatelně aproximuje řadu jiných pravděpodobnostních rozdělení. Třída GaussDistribution poskytuje funkcionalitu generování náhodných čísel splňující toto normální rozdělení. Implementace třídy GaussDistribution je založena na generování čísel uniformního rozdělení a jejich následné Box-Muller transformace na normální rozdělení.

"To, že se náhodná veličina X řídí normálním rozdělením se střední hodnotou μ a rozptylem σ^2 zapisujeme:"[4]

$$X \to N(\mu; \sigma^2)$$ [4]

Hustota pravděpodobnosti:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \mathrm{e}^{-(\frac{x-\mu}{\sqrt{(2)}\sigma})}, -\infty < x < \infty$$

Obrázek 35: Hustota pravděpodobnosti normálního rozdělení [4]

Distribuční funkce:

$$F(x) = \frac{1}{\sigma\sqrt{(2\pi)}} \cdot \int_{-\infty}^{x} \mathrm{e}^{-\frac{t-\mu}{\sqrt{2}\sigma}} \, \mathrm{d}t$$

Obrázek 36: Distribuční funkce normálního rozdělení [4]

6.7 Box-Muller transformace

Pokud máme dvě náhodná nezávislá čísla U_1 a U_2 ze stejné čtvercové hustoty plochy intervalu (0,1) pocházející z uniformního rozdělení, potom X_1 a X_2 vypočítané podle vzorce pro výpočet dvojice náhodných čísel normální rozdělení, tvoří dvojici nezávislých náhodných čísel pocházející z normálního rozdělení se střední hodnotou v bodě nula a rozptylem jedna. [5]

$$X_1 = (-2 \log_e U_1)^{1/2} \cos 2\pi U_2$$
$$X_2 = (-2 \log_e U_1)^{1/2} \sin 2\pi U_2$$

Obrázek 37: Vzorec pro výpočet dvojice náhodných čísel normálního rozdělení [5]

Po získaní náhodného čísla normálního rozdělení se střední hodnotou v bodě nula, je ještě potřeba transformovat tuto hodnotu na normální hodnotu splňující požadovanou střední hodnotu a směrodatnou odchylku podle vzorce pro transformování hodnoty na požadovanou střední hodnotu, kde X_3 je náhodná hodnota normálního rozdělení se střední hodnotou μ a rozptylem σ^2.

Vzorec pro transformování hodnoty na požadovanou střední hodnotu a rozptyl.

$$X_3 = \mu + X_1 * \sigma^2 \tag{3}$$

```
double u1 = 1.0 - mRandom.NextDouble();
double u2 = 1.0 - mRandom.NextDouble();
double x1 = Math.Sqrt(-2.0 * Math.Log(u1)) * Math.Sin(2.0 * Math.PI * u2);
return (int)(mMean + mStdDev * x1);
```

Výpis 2: Funkce realizující Box-Mullerovu transformaci v jazyce C#

Implementace funkce realizující Box-Mullerovu transformaci se skládá z několika kroků. Prvním krokem je vygenerování dvou hodnot v intervalu *(0,1]* . Funkce NextDouble vrací náhodnou hodnotu uniformního rozdělení v rozmezí *[0,1)*, jelikož však v dalším kroku počítáme logaritmus z toho intervalu a nechceme, aby nastala situace, kdy se tento logaritmus počítá z nuly je potřeba vygenerované hodnoty ještě odečíst od hodnoty 1 a zabránit tak vygenerování hodnoty 0. Jakmile vypočítáme hodnotu X_1 pomocí Box-Mullerova postupu, získáváme náhodnou hodnotu normálního rozdělení se střední hodnotou v bodě nula a rozptylem 1. Posledním krokem je transformace této náhodné hodnoty normálního rozdělení na požadovanou náhodnou hodnotu uniformního rozdělení s požadovanou střední hodnotou a určeným rozptylem.

6.8 Exponenciální rozdělení

Posledním možným nastavením distribučního rozdělení pro generování náhodných čísel je exponenciální rozdělení. Exponenciální rozdělení souvisí s Poissonovým rozdělením. Na rozdíl od Poissonova rozdělení, které popisuje počet událostí v nějakém intervalu, Exponenciální rozdělení dobře popisuje dobu čekání do výskytu určitého jevu případně dobu mezi jednotlivými událostmi. Příkladem může být čekaní na nějakou poruchu.

"To, že náhodná veličina X má exponenciální rozdělení s parametrem $\lambda(\lambda > 0)$, budeme zapisovat:"[4] $X \to Exp(\lambda)$

Hustota pravděpodobnosti:

$$f(t) = \begin{cases} \lambda \cdot \mathrm{e}^{-\lambda t}, & t > 0 \\ 0, & t \leqq 0. \end{cases}$$

Obrázek 38: Hustota pravděpodobnosti exponencionálního rozdělení [4]

Distribuční funkce:

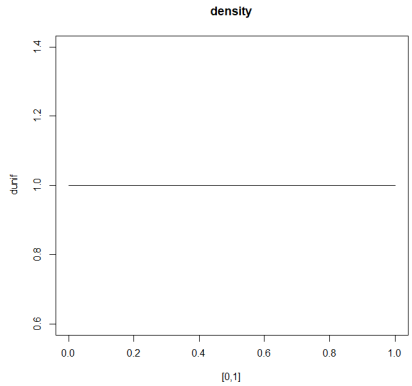
$$F(t) = \begin{cases} 1 - \mathrm{e}^{-\lambda t}, & t > 0 \\ 0, & t \leqq 0. \end{cases}$$

Obrázek 39: Distribuční funkce exponencionálního rozdělení [4]

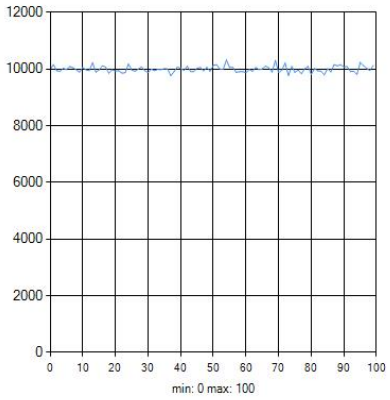
6.9 Testování generování hodnot pro zvolené pravděpodobnostní rozdělení

Řešení obsahuje možnost výběru pravděpodobnostního rozdělení při generování náhodných hodnot. Volba tohoto rozdělení je použita pro náhodný výběr ze zvolené datové sady. Pro ověření správné funkcionality a generování hodnot splňující požadované rozdělení obsahuje řešení funkce

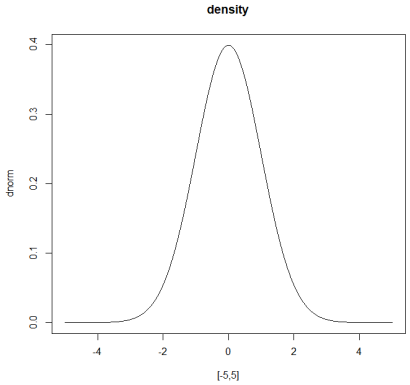
pro vygenerování náhodného zvoleného rozdělení, spočítání četnosti vygenerovaných hodnot, vykreslení distribuční funkce do grafu a následné uložení do souboru. Následně se vytvořené grafy vizuálně porovnají s očekávanými distribučními rozděleními a zhodnotí se, zda daný výstup splňuje požadované vlastnosti.



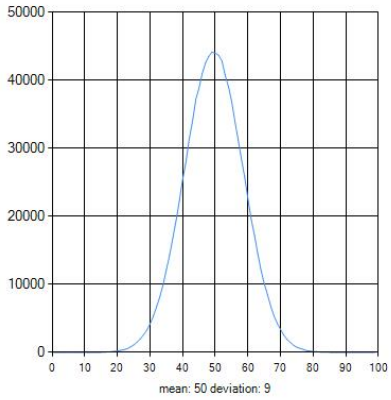
Obrázek 40: Ukázka hustoty uniformního spojitého rozdělení



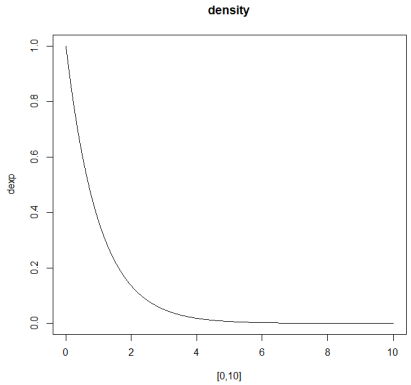
Obrázek 41: Četnost vygenerování jednotlivých jevů pro nastavené uniformní rozdělení



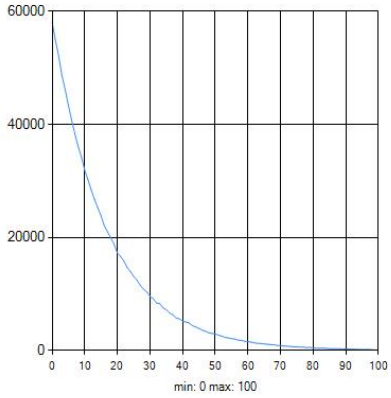
Obrázek 42: Ukázka hustoty normálního rozdělení



Obrázek 43: Četnost vygenerování jednotlivých jevů pro nastavené normální rozdělení



Obrázek 44: Ukázka hustoty exponenciálního rozdělení



Obrázek 45: Četnost vygenerování jednotlivých jevů pro nastavené exponenciální rozdělení

6.10 Dieaharder sada statistických testů

Dieaharder byl vytvořen Robert G. Brownen pro účely testování pseudonáhodných generátorů dat. Sada obsahuje velké množství statistických testů, uzpůsobených přímo pro testování velkého množství vygenerovaných dat. Pro statistické testování je potřeba vzorek s velkým počtem dat, jako dostačující velikost bylo zvoleno $8 * 10^8$ vygenerovaných čísel. Sada obsahuje celkem 113 statistických testů připravených pro testování náhodných generátorů čísel. Výsledkem každého testu je p-hodnota, která vypovídá o pravděpodobnosti, že se daná sekvence skládá ze skutečných náhodných čísel.

Generátor dat obsahuje čtyři rozdílné generátory pseudonáhodných dat, každý s rozdílnou implementací, a tudíž i rozdílnými vlastnostmi.

Pseudo generátory dat:

1. Lineárně kongruentní generátor.
2. Generátor dat třídy Random frameworku .NET pro standartní potřeby.
3. Generátor dat třídy RNGCryptoServiceProvider frameworku .NET pro kryptografické potřeby.
4. Externí hardwarový generátor dat založený na kvantovém generátoru.

Tabulka 9: Výsledek sady testů Dieharder

	LCG	Normal	Crypto	Quantum
Prošel testy	0	25	109	113
Slabý výsledek	0	3	4	0
Neprošlo testy	113	85	0	0
Celkem testů	113	113	113	113

Aplikace generování dat obsahuje různě rychlé generátory pseudonáhodných čísel s rozdílnou kvalitou generovaných dat. Nejrychlejším PRNG je implementace lineárně kongruentního generátoru, tento PRNG je téměř 13 krát rychlejší než kvantový generátor nebo generátor dat pro kryptografické potřeby.

Tabulka 10: Časy generování dat lineárního kongruentního generátoru

LCG	Test 1	Test 2	Test 3	Test 4	Průměr
Doba generování 10^8 vzorků	3.88	3.93	3.86	3.79	3.86
Propustnost [Msample/s]	25.77	25.44	25.90	26.38	25.87

Test byl spuštěn v jednom vláknu procesoru i7-4710HQ a na operačním systému Ubuntu 16.04 za pomocí programu „dieharder version 3.31.1 Copyright 2003 Robert G. Brown“. Namě-

Tabulka 11: Časy generování dat generátoru pro standartní potřeby

Random	Test 1	Test 2	Test 3	Test 4	Průměr
Doba generování 10^8 vzorků	4.11	4.11	4.18	4.2	4.15
Propustnost [Msample/s]	24.33	24.33	23.92	23.80	24.09

Tabulka 12: Časy generování dat generátoru pro kryptografické potřeby

CryptoRandom	Test 1	Test 2	Test 3	Test 4	Průměr
Doba generování 10^8 vzorků	49.18	48.51	50.2	49.2	49.27
Propustnost [Msample/s]	2.03	2.06	1.99	2.03	2.02

Tabulka 13: Časy generování dat kvantového generátoru

Kvantum	Test 1	Test 2	Test 3	Test 4	Průměr
Doba generování 10^8 vzorků	49.93	48.73	47.92	47.76	48.55
Propustnost [Msample/s]	2.00	2.05	2.08	2.09	2.06

řené časy se na různých systémech budou lišit, ale relativní rychlosti mezi jednotlivými generátory zůstanou zachovány.

6.11 Testování vybraných databázových schémat

Poslední zvolenou metodou testování je ověření funkčnosti na komplexních schématech popisující funkci určitého databázového systému. Testování nad takovým schématem odhalí veškeré zbylé chyby, které nebyly odhaleny v průběhu předešlého testování. Dále je možno zanalyzovat úspěšnost navrženého řešení.

Sledované parametry testování:

1. Správnost vytvořených dat
2. Vhodnost vytvořených dat
3. Statistika generování

Správnosti vytvořených dat je myšleno generování dat, která splní požadavky na integritní omezení dané databáze. Pokud by data tyto požadavky nesplňovaly, nebylo by možné vytvořená data vložit do testované databáze. Příkladem vlastnosti, které by zabránily vložení do databáze mohou být špatně vygenerované datové typy pro jednotlivé tabulky, nesplnění vlastností unikátní sekvence pro primární klíče, neexistující reference v cizích klíčích nebo špatné provázání tabulek z důvodu špatné identifikace typu vazeb mezi jednotlivými tabulkami. Existují dva typy vazeb mezi jednotlivými tabulkami. Prvním typem vazby je provázání jedna ku jedné, kde jednomu záznamu tabulky odpovídá jeden a nebo žádný záznam v tabulce. Druhým typem provázání je

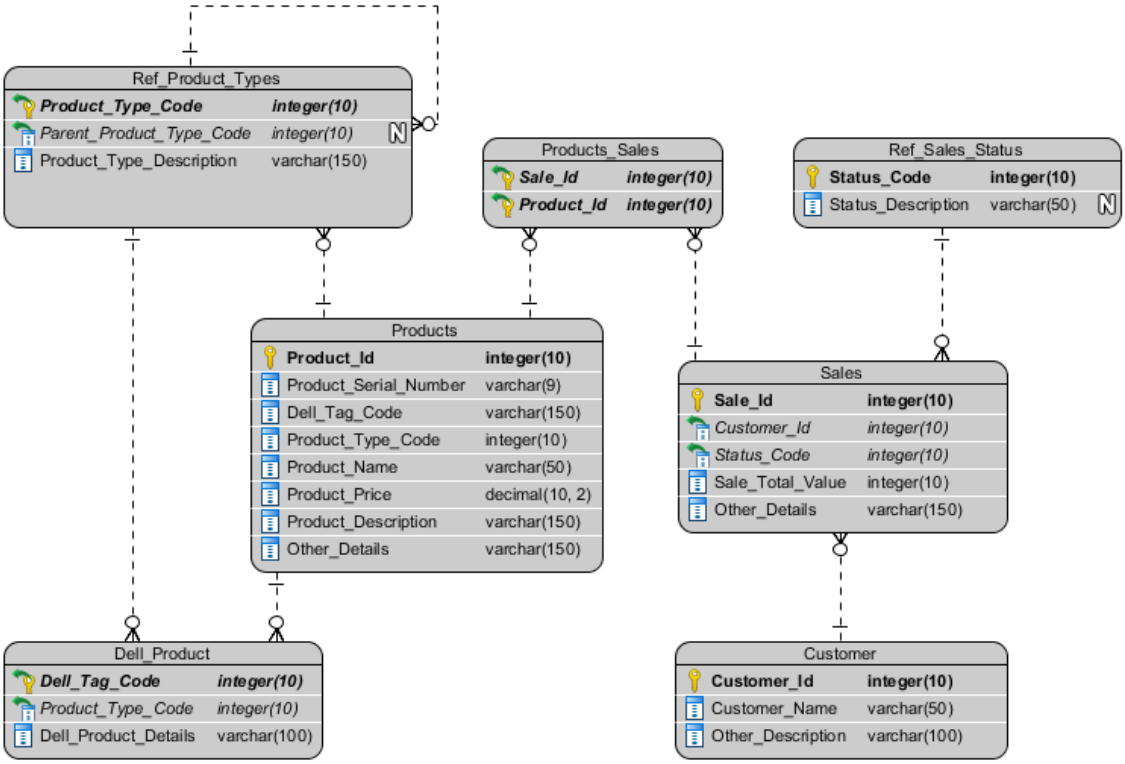
provázání typu jedna ku více, kde jednomu záznamu tabulky odpovídá žádny nebo N záznamů druhé tabulky.

Vhodnostimi vytvořených dat je myšleno, zda datové typy obsahují odpovídající typy datových hodnot. Příkladem může být, zda datový typ integer obsahuje opravdu celočíselné hodnoty, nebo zda datový typ varchar obsahuje znakové řetezce.

Poslední sledovanými parametry jsou statistiky generování. Pozorovanými statistiky generování je celkový počet vygenerovaných dat, doba analyzování zvoleného schématu, doba generování dat, doba ukládání na disk a nebo celkový čas generování a uložení do databázového serveru.

6.11.1 Schéma objednávek Dell produktů

Schéma popisuje databázový systém spravující objednávky na produkty firmy Dell. Schéma obsahuje sedm tabulek vzájemně provázaných tabulek pomocí osmi vazeb. Základní funkcí tohoto schématu je evidence produktů, zákazníků a jejich objednávek. Schéma bylo vytvořeno v nástroji Visual Paradigm za použití návrhu dostupného z webové databáze schémat s názvem Database Answers. [7] Toto schéma bylo rozšířeno o definované datové typy, následně bylo převedeno do SQL skriptu a nahráno do testovacího databázového serveru.



Obrázek 46: Relační diagram schématu prodeje Dell produktů [8]

6.11.2 Schéma ubytování studentů

Testované schéma popisuje správu ubytování studentů v dostupných prostorech. Schéma se skládá ze sedmi tabulek vzájemně provázaných pomocí sedmi vazeb. Toto schéma bylo vytvořeno v nástroji Visual Paradigm za použití návrhu dostupného z webové databáze schémat s názvem Database Answers. [7] Schéma bylo rozšířeno o definované datové typy, následně bylo převedeno do SQL skriptu a nahráno do testovacího databázového serveru.

Obrázek 47: Relační diagram schématu ubytování studentů[9]

s

6.11.3 Zhodnocení generování dat pro testované databázové schémata

Data byla úspěšně vygenerováná a vložena do databázového serveru. Z tohoto faktu lze vyvodit, že data splnila svými vlastnostmi všechna požadovaná integritní omezení. Vizuálním zhodnocením dat byl ověřeno správné přiřazení generovaných hodnot pro dané datové typy tabulek.

Časy ukládání jsou pouze orientační. Doba ukládání závisí na rychlosti připojení do sítě a na aktuálních podmínkách sítě. Doba ukládání na disk se bude lišit pro různé typy disků, aktuální naměřené hodnoty byli získané na hybrydním disku značky Seagate s označením produktu ST1000LM014.

Tabulka 14: Naměřené parametry generování dat schématu prodeje Dell produktů

Naměřeý čas	Popis	Počet generovaných záznamů
746 ms	Načítaní informací o schématu	-
364 ms	Analýza schématu a nastavení generování	-
2 760 ms	Generování	350 000
7 833 ms	Ukládání do databáze	350 000
1 454 ms	Ukládání na disk	350 000
35 208 ms	Generování	3 500 000
89 174 ms	Ukládání do databáze	3 500 000
24 037 ms	Ukládání na disk	3 500 000

Tabulka 15: Naměřené parametry generování dat schématu ubytování studentů

Naměřeý čas	Popis	Počet generovaných záznamů
697 ms	Načítaní informací o schématu	-
390 ms	Analýza schématu a nastavení generování	-
3 770 ms	Generování	350 000
10 340 ms	Ukládání do databáze	350 000
1 112 ms	Ukládání na disk	350 000
39 091 ms	Generování	3 500 000
107 314 ms	Ukládání do databáze	3 500 000
50 412 ms	Ukládání na disk	3 500 000

Ve schématu evidující ubytování studentů se podařilo u tabulky Customer identifikovat bodové vzory a vhodně aplikovat nastavení načítaní dat z datových sad. Nastavení generování dat bylo vybráno pro atributy tabulky s názvy First_Name, Middle_Name, Last_Name a Country.

7 Závěr

V práci se podařilo splnit všechny body předem definovaného zadání. Kromě splnění definovaných bodů se podařilo splnit i další úkoly a dodatečné rozšíření nad rámec definované úkoly. Nad rámec zadání se podařilo realizovat čtyři typy generátorů pseudonáhodných čísel, možnost volby generovaného distribučního rozdělení nebo realizace generování pseudonáhodných čísel pomocí služby kvantového generátoru dat. Dále se pomocí seznamu návrhů schémat studentských databází podařilo navrhnout a otestovat postup pro identifikaci a výběr vzorů v databázovém schématu. Řešení poskytuje poměrně širokou možnost funkcí pro generování dat, uživatelsky přívětivé rozhraní a vhodnou sadu testů a metod testování pro ověření správné celkové funkcionality. Řešení poskytuje nástroj vhodný pro rychlé a snadné generování komplexních dat pro účely naplnění databáze testovacími daty. Navržené řešení několikanásobně zvedá efektivitu vývoje a testování softwaru využívající databázový server pro uložení dat.

Celé řešení je díky správnému návrhu a dodržení metodik správného vývoje snadno rozšiřitelné a udržitelné. Během vývoje bylo dodrženo z velké části metod konfiguračního managmentu. Zdrojový kód byl vyvíjen a spravován pomocí verzovacího systému Git, před samostatným vývojem softwarového díla byl vytvořen prototyp ověřující funkcionalitu a dokazující funkčnost konceptu. Následně po ověření funkčnosti byl vývoj tohoto prototypu ukončen a samostatné softwarové dílo bylo vyvinuto od základu znovu. Během celého vývoje bylo dodrženo testování nejdůležitější funkcionality pomocí unit testů a správně zvolených metodik testování. Kromě testování nově přidané funkcionality během vývoje bylo průběžně ověřována celková stabilita a celková funkčnost systému pomocí regresivního testování. Dále byly také důkladně otestovány jednotlivé generátory pseudonáhodných čísel, porovnány a ověřeny jejich požadované vlastnosti. Během vývoje bylo dodrženo inkrementálního vývoje díla takovým způsobem, že se vyvinula požadovaná funkcionalita. Následně se tato funkcionalita otestovala, zhodnotila a na jejím základě se upravily a rozšířily požadavky na softwarové dílo. Tento proces se několikrát opakovat.

Rěšení je možno dále rozšířit o další vhodná nastavení generování určitých datových typů, rozšířit možnost generování dat pomocí navrženého rozhraní pro další databázové systémy. Dále by bylo vhodné využít dlouhodobého sběru dat využití nástroje pro generování za účelem analýzy využití a identifikace dalších vzorů databázových schémat. Pomocí velkého množství nasbíraných dat, by bylo potom možné prozkoumat možnosti identifikace a nastavení vzorů pomocí nekonvenčních metod, jako jsou například neuronové sítě.

Literatura

[1] Microsoft SQL dokumentace: Data types. *Microsoft* [online]. [cit. 2018-04-24]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql

[2] MENEZES, Alfred, Paul C. VAN OORSCHOT a Scott A. VANSTONE. *Handbook of applied cryptography.* Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 08-493-8523-7.

[3] KNUTH, Donald Ervin. *The art of computer programming.* 3rd ed. Upper Saddle River, NJ: Addison-Wesley, c1998. ISBN 978-0-201-89684-8.

[4] LITSCHMANNOVÁ, Martina. *Vybrané kapitoly z pravděpodobnosti* [online]. Ostrava, 2011 [cit. 2018-04-23]. Dostupné z: http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/vybrane_kapitoly_pravdepodobnost.pdf

[5] BOX, G. E. P. a Mervin MULLER. *A Note on the Generation of Random Normal Deviates* [online]. [cit. 2018-04-23]. Dostupné z: https://projecteuclid.org/download/pdf_1/euclid.aoms/1177706645

[6] Microsoft .NET dokumentace: Data binding. *Microsoft* [online]. 2017 [cit. 2018-04-24]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview

[7] Databáze databázových schémat.*Database Answers: Industry Data Models* [online]. [cit. 2018-04-24]. Dostupné z: http://www.databaseanswers.org/data_models/

[8] Schéma databáze prodeje Dell produktů. *Database Answers: Data Model for Dell Computer Sales* [online]. [cit. 2018-04-24]. Dostupné z: http://www.databaseanswers.org/data_models/del_computer_sales/index.htm

[9] Schéma databáze ubytování studentů. *Database Answers: Student Accomodation* [online]. [cit. 2018-04-24]. Dostupné z: http://www.databaseanswers.org/data_models/student_accommodation/index.htm

[10] FOORD, Michael J. *IronPython in action.* Greenwich, Conn.: Manning, 2009. ISBN 19-339-8833-9.

[11] Návrhový vzor MVVM. Microsoft: Implementing the Model-View-ViewModel Pattern [online]. [cit. 2018-04-24]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff798384.aspx

[12] Implementace výpočtu Levenstainovy vzdálenosti. *Levenshtein Distance, in Three Flavors* [online]. [cit. 2018-04-24]. Dostupné z:

https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm

[13] GAMMA, Erich. Návrh programů pomocí vzorů: stavební kameny objektově orientovaných programů. Praha: Grada, 2003. Moderní programování. ISBN 80-247-0302-5.

A Seznam příloh

1. data-generator - Adresář obsahuje Visual Studio projekt se zdrojovými kódy celé práce
2. images - Adresář obsahuje všechny vytvořené grafy a obrázky, které byly použity v této práci
3. data/Analyze - Adresář obsahující testovací sadu schémat studentských databází
4. data/Dieharder - Adresář obsahující výsledky testovaných generátorů programem Dieharder
5. data/Generation/Duration.xlsm - Soubor s naměřenými časy jednotlivých generátorů
6. data/UniqueGenChart.xlsm - Soubor obsahuje graf počtu porovnání generování unikátních čísel
7. README.txt - Soubor popisující přílohy